CHAPTER 25

Code	*Character*	*Hex*	*Z80 Assembler*	*– after CB*	*– after ED*
214	**VERIFY**	D6	sub N	set 2,(hl)	
215	**BEEP**	D7	rst 16	set 2,a	
216	**CIRCLE**	D8	ret c	set 3,b	
217	**INK**	D9	exx	set 3,c	
218	**PAPER**	DA	jp c,NN	set 3,d	
219	**FLASH**	DB	in a,(N)	set 3,e	
220	**BRIGHT**	DC	call c,NN	set 3,h	
221	**INVERSE**	DD	prefixes instruc-tions using ix	set 3,l	
222	**OVER**	DE	sbc a,N	set 3,(hl)	
223	**OUT**	DF	rst 24	set 3,a	
224	**LPRINT**	E0	ret po	set 4,b	
225	**LLIST**	E1	pop hl	set 4,c	
226	**STOP**	E2	jp po,NN	set 4,d	
227	**READ**	E3	ex (sp),hl	set 4,e	
228	**DATA**	E4	call po,NN	set 4,h	
229	**RESTORE**	E5	push hl	set 4,l	
230	**NEW**	E6	and N	set 4,(hl)	
231	**BORDER**	E7	rst 32	set 4,a	
232	**CONTINUE**	E8	ret pe	set 5,b	
233	**DIM**	E9	jp (hl)	set 5,c	
234	**REM**	EA	jp pe,NN	set 5,d	
235	**FOR**	EB	ex de,hl	set 5,e	
236	**GO TO**	EC	call pe,NN	set 5,h	
237	**GO SUB**	ED		set 5,l	
238	**INPUT**	EE	xor N	set 5,(hl)	
239	**LOAD**	EF	rst 40	set 5,a	
240	**LIST**	F0	ret p	set 6,b	
241	**LET**	F1	pop af	set 6,c	
242	**PAUSE**	F2	jp p,NN	set 6,d	
243	**NEXT**	F3	di	set 6,e	
244	**POKE**	F4	call p,NN	set 6,h	
245	**PRINT**	F5	push af	set 6,l	
246	**PLOT**	F6	or N	set 6,(hl)	
247	**RUN**	F7	rst 48	set 6,a	
248	**SAVE**	F8	ret m	set 7,b	
249	**RANDOMIZE**	F9	ld sp,hl	set 7,c	
250	**IF**	FA	jp m,NN	set 7,d	
251	**CLS**	FB	ei	set 7,e	
252	**DRAW**	FC	call m,NN	set 7,h	
253	**CLEAR**	FD	prefixes instruc-tions using iy	set 7,l	
254	**RETURN**	FE	cp N	set 7,(hl)	
255	**COPY**	FF	rst 56	set 7,a	

The system variables

The bytes in memory from 235521 to 23733 are set aside for specific uses by the system. You can peek them to find out various things about the system, and some of them can be usefully poked. They are listed here with their uses.

These are called *system variables*, and have names, but do not confuse them with the variables used by the BASIC. The computer will not recognize the names as referring to system variables, and they are given solely as mnemonics for we humans.

The abbreviations in column 1 have the following meanings:

X The variables should not be poked because the system might crash.
N Poking the variable will have no lasting effect.

The number in column 1 is the number of bytes in the variable. For two bytes, the first one is the less significant byte – the reverse of what you might expect. So to poke a value **v** to a two-byte variable at address **n**, use

```
POKE n,v-256*INT (v/256)
POKE n+1,INT (v/256)
```

and to peek its value, use the expression

```
PEEK n+256*PEEK (n+1)
```

Notes	*Address*	*Name*	*Contents*
N8	23552	KSTATE	Used in reading the keyboard.
N1	23560	LAST K	Stores newly pressed key.
1	23561	REPDEL	Time (in 50ths of a second – in 60ths of a second in N. America) that a key must be held down before it repeats. This starts off at 35, but you can **POKE** in other values.
1	23562	REPPER	Delay (in 50ths of a second – in 60ths of a second in N. America) between successive repeats of a key held down: initially 5.
N2	23563	DEFADD	Address of arguments of user-defined function if one is being evaluated; otherwise 0.
N1	23565	K DATA	Stores 2nd byte of colour controls entered from keyboard.
N2	23566	TVDATA	Stores bytes of colour, **AT** and **TAB** controls going to television.
X38	23568	STRMS	Addresses of channels attached to streams.
2	23606	CHARS	256 less than address of character set (which starts with space and carries on to the copyright symbol). Normally in ROM, but you can set up your own in RAM and make CHARS point to it.

Notes	Address	Name	Contents
1	236Ø8	RASP	Length of warning buzz.
1	236Ø9	PIP	Length of keyboard click.
1	2361Ø	ERR NR	1 less than the report code. Starts off at 255 (for −1) so **PEEK 2361Ø** gives 255.
X1	23611	FLAGS	Various flags to control the BASIC system.
X1	23612	TV FLAG	Flags associated with the television.
X2	23613	ERR SP	Address of item on machine stack to be used as error return.
N2	23615	LIST SP	Address of return address from automatic listing.
N1	23617	MODE	Specifies K, L, C, E or G cursor.
2	23618	NEWPPC	Line to be jumped to.
1	2362Ø	NSPPC	Statement number in line to be jumped to. Poking first NEWPPC and then NSPPC forces a jump to a specified statement in a line.
2	23621	PPC	Line number of statement currently being executed.
1	23623	SUBPPC	Number within line of statement being executed.
1	23624	BORDCR	Border colour * 8; also contains the attributes normally used for the lower half of the screen.
2	23625	E PPC	Number of current line (with program cursor).
X2	23627	VARS	Address of variables.
N2	23629	DEST	Address of variable in assignment.
X2	23631	CHANS	Address of channel data.
X2	23633	CURCHL	Address of information currently being used for input and output.
X2	23635	PROG	Address of BASIC program.
X2	23637	NXTLIN	Address of next line in program.
X2	23639	DATADD	Address of terminator of last DATA item.
X2	23641	E LINE	Address of command being typed in.
2	23643	K CUR	Address of cursor.
X2	23645	CH ADD	Address of the next character to be interpreted: the character after the argument of **PEEK**, or the **NEWLINE** at the end of a **POKE** statement.
2	23647	X PTR	Address of the character after the **?** marker.
X2	23649	WORKSP	Address of temporary work space.
X2	23651	STKBOT	Address of bottom of calculator stack.
X2	23653	STKEND	Address of start of spare space.
N1	23655	BREG	Calculator's b register.
N2	23656	MEM	Address of area used for calculator's memory. (Usually MEMBOT, but not always.)
1	23658	FLAGS2	More flags.

Code	Character	Hex	Z80 Assembler	– after CB	– after ED
17Ø	**SCREEN$**	AA	xor d	res 5,d	ind
171	**ATTR**	AB	xor e	res 5,e	outd
172	**AT**	AC	xor h	res 5,h	
173	**TAB**	AD	xor l	res 5,l	
174	**VAL$**	AE	xor (hl)	res 5,(hl)	
175	**CODE**	AF	xor a	res 5,a	
176	**VAL**	BØ	or b	res 6,b	ldir
177	**LEN**	B1	or c	res 6,c	cpir
178	**SIN**	B2	or d	res 6,d	inir
179	**COS**	B3	or e	res 6,e	otir
18Ø	**TAN**	B4	or h	res 6,h	
181	**ASN**	B5	or l	res 6,l	
182	**ACS**	B6	or (hl)	res 6,(hl)	
183	**ATN**	B7	or a	res 6,a	
184	**LN**	B8	cp b	res 7,b	lddr
185	**EXP**	B9	cp c	res 7,c	cpdr
186	**INT**	BA	cp d	res 7,d	indr
187	**SQR**	BB	cp e	res 7,e	otdr
188	**SGN**	BC	cp h	res 7,h	
189	**ABS**	BD	cp l	res 7,l	
19Ø	**PEEK**	BE	cp (hl)	res 7,(hl)	
191	**IN**	BF	cp a	res 7,a	
192	**USR**	CØ	ret nz	set Ø,b	
193	**STR$**	C1	pop bc	set Ø,c	
194	**CHR$**	C2	jp nz,NN	set Ø,d	
195	**NOT**	C3	jp NN	set Ø,e	
196	**BIN**	C4	call nz,NN	set Ø,h	
197	**OR**	C5	push bc	set Ø,l	
198	**AND**	C6	add a,N	set Ø,(hl)	
199	**<=**	C7	rst Ø	set Ø,a	
2ØØ	**>=**	C8	ret z	set 1,b	
2Ø1	**<>**	C9	ret	set 1,c	
2Ø2	**LINE**	CA	jp z,NN	set 1,d	
2Ø3	**THEN**	CB		set 1,e	
2Ø4	**TO**	CC	call z,NN	set 1,h	
2Ø5	**STEP**	CD	call NN	set 1,l	
2Ø6	**DEF FN**	CE	adc a,N	set 1,(hl)	
2Ø7	**CAT**	CF	rst 8	set 1,a	
2Ø8	**FORMAT**	DØ	ret nc	set 2,b	
2Ø9	**MOVE**	D1	pop de	set 2,c	
21Ø	**ERASE**	D2	jp nc,NN	set 2,d	
211	**OPEN #**	D3	out (N),a	set 2,e	
212	**CLOSE #**	D4	call nc,NN	set 2,h	
213	**MERGE**	D5	push de	set 2,l	

Code	*Character*		*Hex*	*Z80 Assembler*	*– after CB*	*– after ED*
126	-		7E	ld a,(hl)	bit 7,(hl)	
127	©		7F	ld a,a	bit 7,a	
128	□		80	add a,b	res 0,b	
129	▝		81	add a,c	res 0,c	
130	▘		82	add a,d	res 0,d	
131	▀		83	add a,e	res 0,e	
132	▗		84	add a,h	res 0,h	
133	▐		85	add a,l	res 0,l	
134	▚		86	add a,(hl)	res 0,(hl)	
135	▜		87	add a,a	res 0,a	
136	▖		88	adc a,b	res 1,b	
137	▞		89	adc a,c	res 1,c	
138	▌		8A	adc a,d	res 1,d	
139	▛		8B	adc a,e	res 1,e	
140	▄		8C	adc a,h	res 1,h	
141	▟		8D	adc a,l	res 1,l	
142	▙		8E	adc a,(hl)	res 1,(hl)	
143	█		8F	adc a,a	res 1,a	
144	(a)		90	sub b	res 2,b	
145	(b)		91	sub c	res 2,c	
146	(c)		92	sub d	res 2,d	
147	(d)		93	sub e	res 2,e	
148	(e)		94	sub h	res 2,h	
149	(f)		95	sub l	res 2,l	
150	(g)		96	sub (hl)	res 2,(hl)	
151	(h)		97	sub a	res 2,a	
152	(i)		98	sbc a,b	res 3,b	
153	(j)	user	99	sbc a,c	res 3,c	
154	(k)	graphics	9A	sbc a,d	res 3,d	
155	(l)		9B	sbc a,e	res 3,e	
156	(m)		9C	sbc a,h	res 3,h	
157	(n)		9D	sbc a,l	res 3,l	
158	(o)		9E	sbc a,(hl)	res 3,(hl)	
159	(p)		9F	sbc a,a	res 3,a	
160	(q)		A0	and b	res 4,b	ldi
161	(r)		A1	and c	res 4,c	cpi
162	(s)		A2	and d	res 4,d	ini
163	(t)		A3	and e	res 4,e	outi
164	(u)		A4	and h	res 4,h	
165	**RND**		A5	and l	res 4,l	
166	**INKEY$**		A6	and (hl)	res 4,(hl)	
167	**PI**		A7	and a	res 4,a	
168	**FN**		A8	xor b	res 5,b	ldd
169	**POINT**		A9	xor c	res 5,c	cpd

Notes	*Address*	*Name*	*Contents*
X1	23659	DF SZ	The number of lines (including one blank line) in the lower part of the screen.
2	23660	S TOP	The number of the top program line in automatic listings.
2	23662	OLDPPC	Line number to which **CONTINUE** jumps.
1	23664	OSPCC	Number within line of statement to which **CONTINUE** jumps.
N1	23665	FLAGX	Various flags.
N2	23666	STRLEN	Length of string type destination in assignment.
N2	23668	T ADDR	Address of next item in syntax table (very unlikely to be useful).
2	23670	SEED	The seed for **RND**. This is the variable that is set by **RANDOMIZE**.
3	23672	FRAMES	3 byte (least significant first), frame counter. Incremented every 20ms. See Chapter 18.
2	23675	UDG	Address of 1st user-defined graphic. You can change this for instance to save space by having fewer user-defined graphics.
1	23677	COORDS	x-coordinate of last point plotted.
1	23678		y-coordinate of last point plotted.
1	23679	P POSN	33-column number of printer position.
1	23680	PR CC	Less significant byte of address of next position for **LPRINT** to print at (in printer buffer).
1	23681		Not used.
2	23682	ECHO E	33-column number and 24-line number (in lower half) of end of input buffer.
2	23684	DF CC	Address in display file of **PRINT** position.
2	23686	DFCCL	Like DF CC for lower part of screen.
X1	23688	S POSN	33-column number for **PRINT** position.
X1	23689		24-line number for **PRINT** position.
X2	23690	SPOSNL	Like S POSN for lower part.
1	23692	SCR CT	Counts scrolls: it is always 1 more than the number of scrolls that will be done before stopping with **scroll?**. If you keep poking this with a number bigger than 1 (say 255), the screen will scroll on and on without asking you.
1	23693	ATTR P	Permanent current colours, etc (as set up by colour statements).
1	23694	MASK P	Used for transparent colours, etc. Any bit that is 1 shows that the corresponding attribute bit is taken not from ATTR P, but from what is already on the screen.

Notes	*Address*	*Name*	*Contents*
N1	23695	ATTR T	Temporary current colours, etc (as set up by colour items).
N1	23696	MASK T	Like MASK P, but temporary.
1	23697	P FLAG	More flags.
N30	23698	MEMBOT	Calculator's memory area; used to store numbers that cannot conveniently be put on the calculator stack.
2	23728		Not used.
2	23730	RAMTOP	Address of last byte of BASIC system area.
2	23732	P-RAMT	Address of last byte of physical RAM.

This program tells you the first 22 bytes of the variables area:

```
10 FOR n=0 TO 21
20 PRINT PEEK (PEEK 23627+256*PEEK 23628+n)
30 NEXT n
```

Try to match up the control variable **n** with the descriptions above.
Now change line 20 to

23635 +256*PEEK 23636 +n)

20 PRINT PEEK (~~16509+n)~~

20 PRINT PEEK (23755+n) *

This tells you the first 22 bytes of the program area. Match these up with the program itself.

* FROM 3RD EDITION

Code	*Character*	*Hex*	*Z80 Assembler*	*– after CB*	*– after ED*
82	R	52	ld d,d	bit 2,d	sbc hl,de
83	S	53	ld d,e	bit 2,e	ld (NN),de
84	T	54	ld d,h	bit 2,h	
85	U	55	ld d,l	bit 2,l	
86	V	56	ld d,(hl)	bit 2,(hl)	im 1
87	W	57	ld d,a	bit 2,a	ld a,i
88	X	58	ld e,b	bit 3,b	in e,(c)
89	Y	59	ld e,c	bit 3,c	out (c),e
90	Z	5A	ld e,d	bit 3,d	adc hl,de
91	[	5B	ld e,e	bit 3,e	ld de,(NN)
92	\	5C	ld e,h	bit 3,h	
93	]	5D	ld e,l	bit 3,l	
94	↑	5E	ld e,(hl)	bit 3,(hl)	im 2
95	_	5F	ld e,a	bit 3,a	ld a,r
96	£	60	ld h,b	bit 4,b	in h,(c)
97	a	61	ld h,c	bit 4,c	out (c),h
98	b	62	ld h,d	bit 4,d	sbc hl,hl
99	c	63	ld h,e	bit 4,e	ld (NN),hl
100	d	64	ld h,h	bit 4,h	
101	e	65	ld h,l	bit 4,l	
102	f	66	ld h,(hl)	bit 4,(hl)	
103	g	67	ld h,a	bit 4,a	rrd
104	h	68	ld l,b	bit 5,b	in l,(c)
105	i	69	ld l,c	bit 5,c	out (c),l
106	j	6A	ld l,d	bit 5,d	adc hl,hl
107	k	6B	ld l,e	bit 5,e	ld hl,(NN)
108	l	6C	ld l,h	bit 5,h	
109	m	6D	ld l,l	bit 5,l	
110	n	6E	ld l,(hl)	bit 5,(hl)	
111	o	6F	ld l,a	bit 5,a	rld
112	p	70	ld (hl),b	bit 6,b	in f,(c)
113	q	71	ld (hl),c	bit 6,c	
114	r	72	ld (hl),d	bit 6,d	sbc hl,sp
115	s	73	ld (hl),e	bit 6,e	ld (NN),sp
116	t	74	ld (hl),h	bit 6,h	
117	u	75	ld (hl),l	bit 6,l	
118	v	76	halt	bit 6,(hl)	
119	w	77	ld (hl),a	bit 6,a	
120	x	78	ld a,b	bit 7,b	in a,(c)
121	y	79	ld a,c	bit 7,c	out (c),a
122	z	7A	ld a,d	bit 7,d	adc hl,sp
123	{	7B	ld a,e	bit 7,e	ld sp,(NN)
124	|	7C	ld a,h	bit 7,h	
125	}	7D	ld a,l	bit 7,l	

Code	*Character*	*Hex*	*Z80 Assembler*	*– after CB*	*– after ED*
38	&	26	ld h,N	sla (hl)	
39	'	27	daa	sla a	
40	(	28	jr z,DIS	sra b	
41	)	29	add hl,hl	sra c	
42	*	2A	ld hl,(NN)	sra d	
43	+	2B	dec hl	sra e	
44	,	2C	inc l	sra h	
45	–	2D	dec l	sra l	
46	.	2E	ld l,N	sra (hl)	
47	/	2F	cpl	sra a	
48	0	30	jr nc,DIS		
49	1	31	ld sp,NN		
50	2	32	ld (NN),a		
51	3	33	inc sp		
52	4	34	inc (hl)		
53	5	35	dec (hl)		
54	6	36	ld (hl),N		
55	7	37	scf		
56	8	38	jr c,DIS	srl b	
57	9	39	add hl,sp	srl c	
58	:	3A	ld a,(NN)	srl d	
59	;	3B	dec sp	srl e	
60	<	3C	inc a	srl h	
61	=	3D	dec a	srl l	
62	>	3E	ld a,N	srl (hl)	
63	?	3F	ccf	srl a	
64	@	40	ld b,b	bit 0,b	in b,(c)
65	A	41	ld b,c	bit 0,c	out (c),b
66	B	42	ld b,d	bit 0,d	sbc hl,bc
67	C	43	ld b,e	bit 0,e	ld (NN),bc
68	D	44	ld b,h	bit 0,h	neg
69	E	45	ld b,l	bit 0,l	retn
70	F	46	ld b,(hl)	bit 0,(hl)	im 0
71	G	47	ld b,a	bit 0,a	ld i,a
72	H	48	ld c,b	bit 1,b	in c,(c)
73	I	49	ld c,c	bit 1,c	out (c),c
74	J	4A	ld c,d	bit 1,d	adc hl,bc
75	K	4B	ld c,e	bit 1,e	ld bc,(NN)
76	L	4C	ld c,h	bit 1,h	
77	M	4D	ld c,l	bit 1,l	reti
78	N	4E	ld c,(hl)	bit 1,(hl)	
79	O	4F	ld c,a	bit 1,a	ld r,a
80	P	50	ld d,b	bit 2,b	in d,(c)
81	Q	51	ld d,c	bit 2,c	out (c),d

CHAPTER 26

The character set

This is the complete Spectrum character set, with codes in decimal and hex. If one imagines the codes as being Z80 machine code instructions, then the right hand columns give the corresponding assembly language mnemonics. As you are probably aware if you understand these things, certain Z80 instructions are compounds starting with CBh or EDh; the two right hand columns give these.

Code	*Character*	*Hex*	*Z80 Assembler*	*– after CB*	*– after ED*
0		00	nop	rlc b	
1		01	ld bc,NN	rlc c	
2	not used	02	ld (bc),a	rlc d	
3		03	inc bc	rlc e	
4		04	inc b	rlc h	
5		05	dec b	rlc l	
6	**PRINT** comma	06	ld b,N	rlc (hl)	
7	**EDIT**	07	rlca	rlc a	
8	cursor left	08	ex af,af'	rrc b	
9	cursor right	09	add hl,bc	rrc c	
10	cursor down	0A	ld a,(bc)	rrc d	
11	cursor up	0B	dec bc	rrc e	
12	**DELETE**	0C	inc c	rrc h	
13	**ENTER**	0D	dec c	rrc l	
14	number	0E	ld c,N	rrc (hl)	
15	not used	0F	rrca	rrc a	
16	**INK** control	10	djnz DIS	rl b	
17	**PAPER** control	11	ld de,NN	rl c	
18	**FLASH** control	12	ld (de),a	rl d	
19	**BRIGHT** control	13	inc de	rl e	
20	**INVERSE** control	14	inc d	rl h	
21	**OVER** control	15	dec d	rl l	
22	**AT** control	16	ld d,N	rl (hl)	
23	**TAB** control	17	rla	rl a	
24		18	jr DIS	rr b	
25		19	add hl,de	rr c	
26		1A	ld a,(de)	rr d	
27	not used	1B	dec de	rr e	
28		1C	inc e	rr h	
29		1D	dec e	rr l	
30		1E	ld e,N	rr (hl)	
31		1F	rra	rr a	
32	space	20	jr nz,DIS	sla b	
33	!	21	ld hl,NN	sla c	
34	"	22	ld (NN),hl	sla d	
35	#	23	inc hl	sla e	
36	$	24	inc h	sla h	
37	%	25	dec h	sla l	

Using machine code

Summary
USR with numeric argument

This chapter is written for those who understand *Z80 machine code,* the set of instructions that the Z80 processor chip uses. If you do not, but would like to, there are plenty of books about it. You want to get one called something along the lines of 'Z80 Machine code (or assembly language) for the absolute beginner', and if it mentions the Spectrum, so much the better.

These programs are normally written in *assembly language,* which, although cryptic, are not too difficult to understand with practice. (You can see the assembly language instructions in Appendix A.) However, to run them on the computer you need to code the program into a sequence of bytes – in this form it is called *machine code.* This translation is usually done by the computer itself, using a program called an *assembler.* There is no assembler built in to the Spectrum, but you may well be able to buy one on cassette. Failing that, you will have to do the translation yourself, provided that the program is not too long.

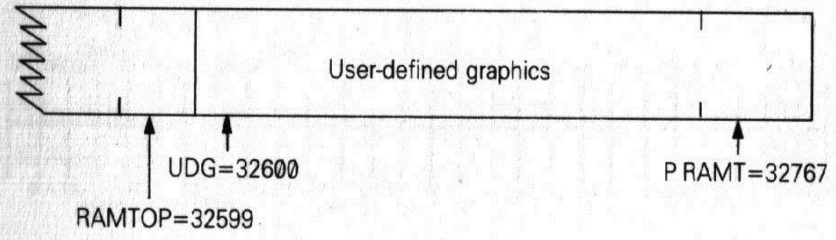
Let's take as an example the program

```
ld bc, 99
ret
```

which loads the bc register pair with 99. This translates into the four machine code bytes 1, 99, 0 (for ld bc, 99) and 201 (for ret). (If you look up 1 and 201 in Appendix A, you will find ld bc, NN – where NN stands for any two-byte number – and ret.)

When you have got your machine code program, the next step is to get it into the computer. (An assembler would probably do this automatically.) You need to decide whereabouts in memory to put it, and the best thing is to make extra space for it between the BASIC area and the user-defined graphics.

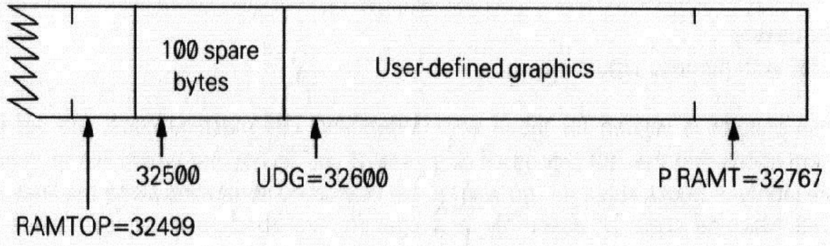
Suppose, for instance, that you have a 16K Spectrum. To start off with, the top end of RAM has



If you type

```
CLEAR 32499
```

this will give you a space of 100 (for good measure) bytes starting at address 32500.



To put in the machine code program, you would run a BASIC program something like

```
10 LET a=32500
20 READ n: POKE a,n
30 LET a=a+1: GO TO 20
40 DATA 1,99,0,201
```

(This will stop with report **E Out of DATA** when it has filled in the four bytes you specified.)

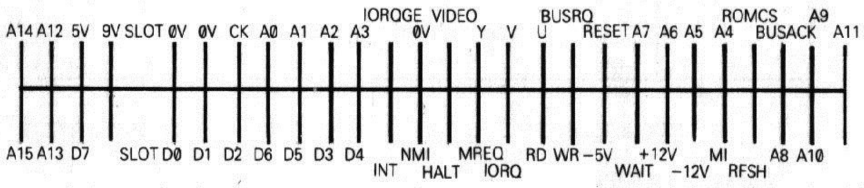
To run the machine code, you use the function **USR** – but this time with a numeric argument, the starting address. Its result is the value of the bc register on return from the machine code program, so if you do

```
PRINT USR 32500
```

you get the answer 99.

The return address to the BASIC is stacked in the usual way, so return is by a Z80 ret instruction. You should not use the iy and i registers in a machine code routine.

The control, data and address busses are all exposed at the back of the Spectrum, so you can do almost anything with a Spectrum that you can with a Z80. Sometimes, though, the Spectrum hardware might get in the way. Here is a diagram of the exposed connections at the back:


A14 A12 5V 9V SLOT 0V 0V CK A0 A1 A2 A3 IORQGE VIDEO 0V Y V BUSRQ U RESET A7 A6 A5 ROMCS A4 BUSACK A9 A11
A15 A13 D7 SLOT D0 D1 D2 D6 D5 D3 D4 INT NMI HALT MREQ IORQ RD WR −5V WAIT +12V −12V MI RFSH A8 A10

You can save your machine code program easily enough with

```
SAVE "some name" CODE 32500,4
```

On the face of it, there is no way of saving it so that when loaded it automatically runs itself, but you can get round this by using a BASIC program.

```
10 LOAD "" CODE 32500,4
20 PRINT USR 32500
```

Do first

```
SAVE "some name" LINE
```

and then

```
SAVE "xxxx" CODE 32500,4
LOAD "some name"
```

will then load and automatically run the BASIC program, and the BASIC program will load and run the machine code.

Notes

1. RST 38h calls Keyboard interpreter routine (op codes at 0056h tables at 0205h)
2 Sound routine entry point 03B5h. HL set for pitch (high value = low pitch), DE set for length of note (high value = long note)
3 COPY routine held at 0EACh (no parameters). Can be entered at 0EB2h but interrupts must be disabled, B set to no of scans (8 scans = 1 print line) and HL set to point to first line to be printed.
4 One high res line can be printed with routine at 0EF4h using HL to point to line. Before calling disable interupts, after return switch printer off (OUT FB A) and enable interupts.