Notes on Representing Multi-Byte PS/2 Keyboard Scan Codes as Single-Byte ASCII-based Characters

Michael P. Frank
Wednesday, November 29, 2006

Version 4 of this document presents an improved scheme for the decoding of byte sequences, with a greatly simplified FSM design.

Version 5 rearranges some of the ASCII coding of keys so that more of the commonly-encountered punctuation characters are used to represent associated keypresses.

Introduction

The standard PS/2-compatible keyboards used with most PCs in the United States generate multiple-byte sequences to represent various key-press ("make") and key-release ("break") events. For designing custom digital systems that can accept input from such keyboards, it is convenient if the multi-byte make/break sequences are pre-translated into a convenient single-byte format that is easy to interpret using a simple combinational circuit. In this document, we present a proposed mapping from multi-byte sequences into single-byte codes based on the standard US 7-bit ASCII character set. We also summarize the design of a simple state machine that can accomplish this translation.

List of Codes

In the below table I am inventing a mapping from key make and break events on a standard 104-key PS/2 US keyboard (scan code set 2) to 8-bit characters that correspond, in as many cases as possible, to ASCII characters that are related in some way to the key pressed. The break codes are the same as the make codes except with bit 7 turned on.

Raw input byte sequences for most standard US PS/2 keyboards			**My Custom ASCII-based Encoding**			
Key	**Make code**	**Break code**	**ASCII make char**	**ASCII make hex**	**ASCII break hex**	**Notes**
A	1C	F0,1C	A	41	C1	Use uppercase for letter keys
B	32	F0,32	B	42	C2	
C	21	F0,21	C	43	C3	
D	23	F0,23	D	44	C4	
E	24	F0,24	E	45	C5	
F	2B	F0,2B	F	46	C6	
G	34	F0,34	G	47	C7	
H	33	F0,33	H	48	C8	
I	43	F0,43	I	49	C9	
J	3B	F0,3B	J	4A	CA	
K	42	F0,42	K	4B	CB	
L	4B	F0,4B	L	4C	CC	
M	3A	F0,3A	M	4D	CD	

Raw input byte sequences for most standard US PS/2 keyboards			My Custom ASCII-based Encoding			
Key	Make code	Break code	ASCII make char	ASCII make hex	ASCII break hex	Notes
N	31	F0,31	N	4E	CE	
O	44	F0,44	O	4F	CF	
P	4D	F0,4D	P	50	D0	
Q	15	F0,15	Q	51	D1	
R	2D	F0,2D	R	52	D2	
S	1B	F0,1B	S	53	D3	
T	2C	F0,2C	T	54	D4	
U	3C	F0,3C	U	55	D5	
V	2A	F0,2A	V	56	D6	
W	1D	F0,1D	W	57	D7	
X	22	F0,22	X	58	D8	
Y	35	F0,35	Y	59	D9	
Z	1A	F0,1A	Z	5A	DA	
) 0	45	F0,45	)	29	A9	Use punctuation to disting. these fr. the #s on keypad
! 1	16	F0,16	!	21	A1	
@ 2	1E	F0,1E	@	40	C0	
# 3	26	F0,26	#	23	A3	
$ 4	25	F0,25	$	24	A4	
% 5	2E	F0,2E	%	25	A5	
^ 6	36	F0,36	^	5E	DE	
& 7	3D	F0,3D	&	26	A6	
* 8	3E	F0,3E	*	2A	AA	
(9	46	F0,46	(	28	A8	
~ `	0E	F0,0E	`	60	E0	Unshifted punctuation char
_ -	4E	F0,4E	_	5F	DF	To distinguish from keypad -
+ =	55	F0,55	=	3D	BD	Unshifted punctuation char
| \	5D	F0,5D	\	5C	DC	Unshifted punctuation char
Backsp	66	F0,66	BS ⌫	08	88	Standard ASCII control
Space	29	F0,29	SP ␣	20	A0	Standard ASCII control
Tab	0D	F0,0D	HT ↦	09	89	Standard ASCII control

Raw input byte sequences for most standard US PS/2 keyboards			My Custom ASCII-based Encoding			
Key	Make code	Break code	ASCII make char	ASCII make hex	ASCII break hex	Notes
CapsLk	58	F0,58	DC2 ⊛	12	92	"Special mode" control
L Shft	12	F0,12	SO ⊙	0E	8E	"Outermost" shift key
L Ctrl	14	F0,14	{	7B	FB	Left & right control: { , }
L GUI	E0,1F	E0,F0,1F	v	76	F6	Windows keys: v & w
L Alt	11	F0,11	a	61	E1	Left & right alt: a & b
R Shft	59	F0,59	SI ⊗	0F	8F	"Innermost" shift key
R Ctrl	E0,14	E0,F0,14	}	7D	FD	Left & right control: { , }
R GUI	E0,27	E0,F0,27	w	77	F7	Windows keys: v & w
R Alt	E0,11	E0,F0,11	b	62	E2	Left & right alt: a & b
Menu	E0,2F	E0,F0,2F	m	6D	ED	m is for menu
Enter	5A	F0,5A	CR ↲	0D	8D	Line feed left implicit
Esc	76	F0,76	ESC ⇑	1B	9B	Standard ASCII control char
F1	05	F0,05	ENQ ⍰	05	85	Help function, like ENQ
F2	06	F0,06	GS ⌘	1D	9D	"two"
F3	04	F0,04	ACK ☑	04	84	Arbitrary control char. Was "f" for Find File function
F4	0C	F0,0C	EOT ↯	04	84	Alt-F4 terminates app
F5	03	F0,03	DC1 ◎	11	91	Refresh: Reset, turn on, XON
F6	0B	F0,0B	BEL ☎	07	87	Arbitrary control char. Was "c" for Cycle cursor around
F7	83	F0,83	CAN ✖	18	98	Cancel misspelling
F8	0A	F0,0A	DC4 ⊖	13	93	Arbitrary control char. Was "e" for Extend selection in Word
F9	01	F0,01	US ∫	1F	9F	Update Selected fields
F10	09	F0,09	NAK ⊠	15	15	Arbitrary control char. Was backtick, for menu bar in upper-left corner
F11	78	F0,78	FS ⌘	1C	9C	Full Screen mode toggle
F12	07	F0,07	ETB ⊡	17	97	End work, Save As…
PrtScr SysRq	E0,12, E0,7C	E0,F0,7C, E0,F0,12	DLE ⌥	10	90	Escape to OS
ScrLck	7E	F0,7E	SYN ⍉	16	96	Used to pause output
Pause Break[1]	E1,14,77	E1,F0,14, F0,77	DC3 ⊖	13	93	Pause function

[1] Note: The break code for the Pause/Break key is always sent immediately following its make code, without waiting for the key to be released. No code is sent when the key is actually released.

Raw input byte sequences for most standard US PS/2 keyboards			My Custom ASCII-based Encoding			
Key	Make code	Break code	ASCII make char	ASCII make hex	ASCII break hex	Notes
{ [	54	F0,54	]	5B	DB	Unshifted punctuation char
Insert	E0,70	E0,F0,70	STX ⊥	02	82	Insert start of new text here
Home	E0,6C	E0,F0,6C	SOH ⌜	01	81	Start of document
Pg Up	E0,7D	E0,F0,7D	VT ↧	0B	8B	VT can move backwards
Delete	E0,71	E0,F0,71	DEL ⌦	7F	FF	Forwards delete character
End	E0,69	E0,F0,69	ETX ⌟	03	83	End of document
Pg Dn	E0,7A	E0,F0,7A	FF ↡	0C	8C	Go to start of next page
U arr	E0,75	E0,F0,75	u	75	F5	u is for up
L arr	E0,6B	E0,F0,6B	l	6C	EC	l is for left
R arr	E0,74	E0,F0,72	r	72	F2	r is for right
D arr	E0,72	E0,F0,74	d	64	E4	d is for down
NumLk	77	F0,77	n	6E	EE	n is for num lock
KP /	E0,4A	E0,F0,4A	/	2F	AF	Label on keypad key
KP *	7C	F0,7C	x	78	F8	x means multiply
KP -	7B	F0,7B	-	2D	AD	Label on keypad key
KP +	79	F0,79	p	70	F0	p means plus
KP Ent	E0,5A	E0,F0,5A	LF ↓	0A	8A	Let this enter do a LF
KP .	71	F0,71	.	2E	AE	Label on keypad key
KP 0	70	F0,70	0	30	B0	Label on keypad key
KP 1	69	F0,69	1	31	B1	Label on keypad key
KP 2	72	F0,72	2	32	B2	Label on keypad key
KP 3	7A	F0,7A	3	33	B3	Label on keypad key
KP 4	6B	F0,6B	4	34	B4	Label on keypad key
KP 5	73	F0,73	5	35	B5	Label on keypad key
KP 6	74	F0,74	6	36	B6	Label on keypad key
KP 7	6C	F0,6C	7	37	B7	Label on keypad key
KP 8	75	F0,75	8	38	B8	Label on keypad key
KP 9	7D	F0,7D	9	39	B9	Label on keypad key
}]	5B	F0,5B	]	5D	DD	Unshifted punctuation char
: ;	4C	F0,4C	:	3A	BA	Unshifted punctuation char
" '	52	F0,52	'	60	E0	Unshifted punctuation char
< ,	41	F0,41	,	2C	AC	Unshifted punctuation char
> .	49	F0,49	>	3E	BE	To dist. fr. keypad "."
? /	4A	F0,4A	?	3F	BF	To dist. fr. keypad "/"

Here's the same table again, this time sorted by the raw make codes (column 2).

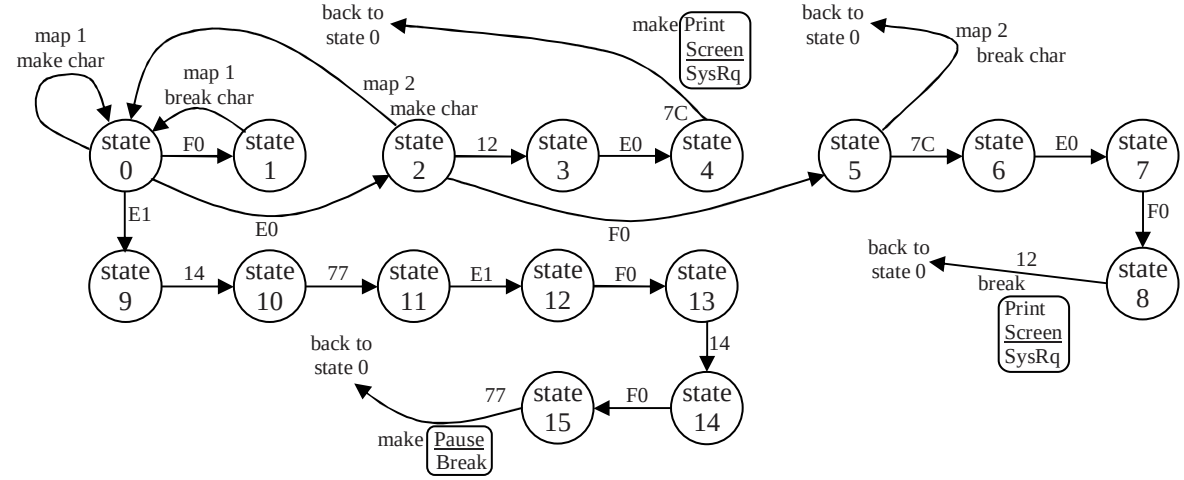
Key	Make code	Break code	ASCII make char	ASCII make hex	ASCII break hex
F9	01	F0,01	US ∫	1F	9F
F5	03	F0,03	DC1 ◎	11	91
F3	04	F0,04	ACK ☑	06	86
F1	05	F0,05	ENQ ⍰	05	85
F2	06	F0,06	GS ⨋	1D	9D
F12	07	F0,07	ETB ⊡	17	97
F10	09	F0,09	NAK ☒	15	95
F8	0A	F0,0A	DC4 ⊖	13	93
F6	0B	F0,0B	BEL ☎	07	87
F4	0C	F0,0C	EOT ↯	04	84
Tab	0D	F0,0D	HT ↦	09	89
~ `	0E	F0,0E	`	60	E0
L Alt	11	F0,11	a	61	E1
L Shft	12	F0,12	SO ⊙	0E	8E
L Ctrl	14	F0,14	{	7B	FB
Q	15	F0,15	Q	51	D1
! 1	16	F0,16	!	21	A1
Z	1A	F0,1A	Z	5A	DA
S	1B	F0,1B	S	53	D3
A	1C	F0,1C	A	41	C1
W	1D	F0,1D	W	57	D7
@ 2	1E	F0,1E	@	40	C0
C	21	F0,21	C	43	C3
X	22	F0,22	X	58	D8
D	23	F0,23	D	44	C4
E	24	F0,24	E	45	C5
$ 4	25	F0,25	$	24	A4
# 3	26	F0,26	#	23	A3
Space	29	F0,29	SP ␣	20	A0
V	2A	F0,2A	V	56	D6
F	2B	F0,2B	F	46	C6
T	2C	F0,2C	T	54	D4
R	2D	F0,2D	R	52	D2
% 5	2E	F0,2E	%	25	A5

Key	Make code	Break code	ASCII make char	ASCII make hex	ASCII break hex
N	31	F0,31	N	4E	CE
B	32	F0,32	B	42	C2
H	33	F0,33	H	48	C8
G	34	F0,34	G	47	C7
Y	35	F0,35	Y	59	D9
^ 6	36	F0,36	^	5E	DE
M	3A	F0,3A	M	4D	CD
J	3B	F0,3B	J	4A	CA
U	3C	F0,3C	U	55	D5
& 7	3D	F0,3D	&	26	A6
* 8	3E	F0,3E	*	2A	AA
< ,	41	F0,41	,	2C	AC
K	42	F0,42	K	4B	CB
I	43	F0,43	I	49	C9
O	44	F0,44	O	4F	CF
) 0	45	F0,45	)	29	A9
(9	46	F0,46	(	28	A8
> .	49	F0,49	>	3E	BE
? /	4A	F0,4A	?	3F	BF
L	4B	F0,4B	L	4C	CC
: ;	4C	F0,4C	;	3B	BB
P	4D	F0,4D	P	50	D0
_ -	4E	F0,4E	_	5F	DF
“ ‘	52	F0,52	’	27	A7
{ [	54	F0,54	[	5B	DB
+ =	55	F0,55	=	3D	BD
CapsLk	58	F0,58	DC2 ⊛	12	92
R Shft	59	F0,59	SI ⊗	0F	8F
Enter	5A	F0,5A	CR ↲	0D	8D
}]	5B	F0,5B	]	5D	DD
| \	5D	F0,5D	/	5C	DC
Backsp	66	F0,66	BS ⌫	08	88
KP 1	69	F0,69	1	31	B1
KP 4	6B	F0,6B	4	34	B4
KP 7	6C	F0,6C	7	37	B7
KP 0	70	F0,70	0	30	B0

Key	Make code	Break code	ASCII make char	ASCII make hex	ASCII break hex
KP .	71	F0,71	.	2E	AE
KP 2	72	F0,72	2	32	B2
KP 5	73	F0,73	5	35	B5
KP 6	74	F0,74	6	36	B6
KP 8	75	F0,75	8	38	B8
Esc	76	F0,76	ESC ⇑	1B	9B
NumLk	77	F0,77	n	6E	EE
F11	78	F0,78	FS ⌗	1C	9C
KP +	79	F0,79	+	2B	AB
KP 3	7A	F0,7A	3	33	B3
KP -	7B	F0,7B	-	2D	AD
KP *	7C	F0,7C	x	78	F8
KP 9	7D	F0,7D	9	39	B9
ScrLck	7E	F0,7E	SYN ⍉	16	96
F7	83	F0,83	CAN ✖	18	98
R Alt	E0,11	E0,F0,11	b	62	E2
PrtScr SysRq	E0,12, E0,7C	E0,F0,7C, E0,F0,12	DLE ⌥	10	90
R Ctrl	E0,14	E0,F0,14	}	7D	FD
L GUI	E0,1F	E0,F0,1F	v	76	F6
R GUI	E0,27	E0,F0,27	w	77	F7
Menu	E0,2F	E0,F0,2F	m	6D	ED
KP /	E0,4A	E0,F0,4A	/	2F	AF
KP Ent	E0,5A	E0,F0,5A	LF ↓	0A	8A
End	E0,69	E0,F0,69	ETX ⌟	03	83
L arr	E0,6B	E0,F0,6B	l	6C	EC
Home	E0,6C	E0,F0,6C	SOH ⌜	01	81
Insert	E0,70	E0,F0,70	STX ⊥	02	82
Delete	E0,71	E0,F0,71	DEL ⌦	7F	FF
R arr	E0,74	E0,F0,72	r	72	F2
D arr	E0,72	E0,F0,74	d	64	E4
U arr	E0,75	E0,F0,75	u	75	F5
Pg Dn	E0,7A	E0,F0,7A	FF ⇟	0C	8C
Pg Up	E0,7D	E0,F0,7D	VT ↧	0B	8B
Pause Break	E1,14, 77	E1,F0,14, F0,77	DC3 ⊖	13	93

State Machine Design

Here is a state diagram for a machine to decode the above sequences:



This table shows the "expected" next byte in state sequences:

State:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Byte:	-	-	12	E0	7C	7C	E0	F0	12	14	77	E1	F0	14	F0	77

The bold lines indicate that if the expected next byte is seen when in the state immediately to the left, a make or break code for some special key is output and then we return to state 0 instead of going to the next state.

We can simplify things slightly and get things working properly for the cases NumLock+ExtendedKeypad and Shift+PrtScSysRq if we just treat the sequence E0,12 as a character in its own right, "Imaginary Extended Shift" and represent it with ASCII 00 (NUL, ∅). With this, PrintScreen/SysRq becomes an ordinary map 2 byte sequence (E0, 7C) except that the imaginary extended shift may sometimes come before it. Meanwhile, we also eliminate states 6, 7, and 8. This leaves us with Pause/Break. However, we can simplify its processing by treating it as two events: first Make Pause/Break, then immediately followed by Break Pause/Break. These can each be processed by their own branches. In practice, they will always go together, but so what. Otherwise, software will have to specially deal with the fact that Pause/Break doesn't have a separate break code. So this simplifies the design. We can simplify even further by treating the E1 extension code as identical to the E0 code for primary decoding purposes, so that Pause/Break becomes an ordinary Map 2 character as well. The only special provision here is that we need to suppress the ordinary output for Control which would have been generated by an E0 14 sequence, and stay in the extended-sequence state after the 14 (control) and use it to process the 77 also. This suggests a 3-bit code with bits:

E0 – Received an E0 code, go into extended sequence for map 2
E1 – Received an E1 code, go into extended sequence for map 2, and stay longer
F0 – Received an F0 code, generating a break character

See the section "Simplified Logic for Key Decoder Module" at the end of this document.

Code Pages

This section shows convenient 2-dimensional charts to represent the lookup tables that will be required in the key decoder.

Map 1, from 1st byte of sequence to output ASCII make code or special state:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		1F		11	06	05	1D	17		15	13	07	04	09	60	
1		61	0E		7B	51	21				5A	53	41	57	40	
2		43	58	44	45	24	23			20	56	46	54	52	25	
3		4E	42	48	47	59	5E				4D	4A	55	26	2A	
4		2C	4B	49	4F	29	28			3E	3F	4C	3B	50	5F	
5			27		5B	3D			12	0F	0D	5D		5C		
6							08			31		34	37			
7	30	2E	32	35	36	38	1B	6E	1C	2B	33	2D	78	39	16	
8				18												
9																
A																
B																
C																
D																
E	**st2**	**st9**														
F	**st1**															

This view of map 1 is like the previous chart but shows instead how the keys are labeled:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		F9		F5	F3	F1	F2	F12		F10	F8	F6	F4	Tab	~ `	
1		Left Alt	Left Shift		Left Ctrl	Q q	! 1				Z z	S s	A a	W w	@ 2	
2		C c	X x	D d	E e	$ 4	# 3			Space	V v	F f	T t	R r	% 5	
3		N n	B b	H h	G g	Y y	^ 6				M m	J j	U u	& 7	* 8	
4		< ,	K k	I i	O o	) 0	(9			> .	? /	L l	: ;	P p	_ -	
5			“ ’		{ [	+ =			Caps Lock	Right Shift	Enter	}]		| \		
6							Back space			KP 1		KP 4	KP 7			
7	KP 0	KP .	KP 2	KP 5	KP 6	KP 8	Esc	Num Lock	F11	KP +	KP 3	KP -	KP *	KP 9	Scroll Lock	
8				F7												
9																
A																
B																
C																
D																
E	st2	st9														
F	st1															

Some things to note:

- All valid single-byte make codes are only 7 bits long (that is, bit 7 = 0) except for the code for function key F7 (hex 83).
- All byte values that initiate extended byte sequences begin with the high bits 111.

Finally, this third version of map 1 shows the glyphs for the ASCII characters that I am choosing to represent each key:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		∫		◎	☑	⍰	⋕	▣		☒	⊖	☎	↯	↦	`	
1		a	⊙		{	Q	!				Z	S	A	W	@	
2		C	X	D	E	$	#			␣	V	F	T	R	%	
3		N	B	H	G	Y	^				M	J	U	&	*	
4		,	K	I	O	)	(			>	?	L	;	P	_	
5			’		[	=			⊛	⊗	↲	]		\		
6							⌫			1		4	7			
7	0	.	2	5	6	8	⇑	n	⋕	+	3	-	x	9	⌚	
8				✖												
9																
A																
B																
C																
D																
E	st2	st9														
F	st1															

This next code chart is map 2, for the 2nd byte of 2-byte make sequences starting with E0 (from state 2):

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1		**Right Alt**	**st3 / ExSh**		**Right Ctrl**											**Left GUI**
2								**Right GUI**								**Menu**
3																
4											**KP /**					
5											**KP Enter**					
6										**End**		←	**Home**			
7	**Ins-ert**	**Del-ete**	↓		→	↑		**Pause Break**			**Page Down**		**PrSc SyRq**	**Page Up**		
8																
9																
A																
B																
C																
D																
E																
F	**st5**															

Notes:

- Byte sequence E0,12 (` ⊛) can be considered to represent an imaginary "extended shift" key. Notice this sequence consists of the extended-keymap escape code 12, followed the ordinary Left Shift key code 12. This special "extended shift" character begins the code sequence for the Print Screen / SysRq key, and when the Num Lock key is active, it also prefixes the code sequences for all of the extended-keypad keys (Insert, Delete, Home, End, Page Up, Page Down, and the ↑, ↓, ←, → arrow keys).
- The undecorated byte sequence E0,7C is generated by the Print Screen / SysRq key when a shift key is held down. If shift is not being held down, this sequence is further prefixed by E0,12, the imaginary extended-shift key.
- Pause/Break actually starts with E1, not E0, but we can still place it on this map.

Next view of map 2 shows the chosen ASCII characters for those:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1		b	**st3** ∅		}											v
2								w								m
3																
4											/					
5											↓					
6										⌟		l	⌜			
7	⊥	⌦	d		r	u		⊜			⇟		⌥	↧		
8																
9																
A																
B																
C																
D																
E																
F	**st5**															

Same thing, but now with the hex codes:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1		62	**st3** 00		7D											76
2								77								6D
3																
4											2F					
5											0A					
6										03		6C	01			
7	02	7F	64		72	75		14			0C		10	0B		
8																
9																
A																
B																
C																
D																
E																
F	**st5**															

Some things to note about Map 2:

- The codes for right-Alt and right-Ctrl are the same as left-Alt and left-Ctrl but with E0 prefixed.
- The menu, left GUI, and right GUI codes are distinct from any codes on map 1.
- The keypad "/" code is the same as the code for the ?/ key but with E0 prefixed.
- The codes for Insert, Delete, Home, End, Page Up, Page Down, and the four arrow keys (Up, Dn, Lf, Rt) are the same as those for the correspondingly labeled keypad keys (resp. 0, ., 7, 1, 9, 3, 8, 2, 4, 6), except with E0 (`) prefixed.
    - Depending on the state of the keyboard's built-in Num Lock flag, alternate make/break sequences may be generated for these keys instead! This behavior is not documented, but I have reverse-engineered it below.

- If “Num Lock” has been pressed an odd number of times since the last reset, the make codes for the extended-keypad keys (Ins, Del, Home End, PgUp, PgDn, Up, Dn, Lf, Rt) are prefixed by E0,12 (` ⊛) and the break codes are suffixed by E0,F0,12 (` p ⊛). This would correspond to make and break codes for an extended-shift character.
- This same extended-shift character also makes an appearance in the codes for PrSc/SysRq: make code (` ⊛ ` |), break code (` p | ` p ⊛). Interestingly, if either of the ordinary shift keys is held down while PrSc/SysRq is pressed, the extended-shift press/release codes get taken away, and PrSc/SysRq just generates the codes make (` |); break (` p ⊛), which suggests that one can just ignore the extended-shift press and release events in all cases, and a simpler test for PrSc/SysRq and the extended-keypad keys would be to just look for their press-release codes sans the extended-shift character.
- The Pause/Break make sequence looks like a make-break sequence consisting of:
    - make code: 2nd ary extended shift E1 (a) followed by make-LCtrl, make-NumLock sequence 14, 77 (⊖ w).
    - break code: 2nd ary extended shift E1 (a) followed by break-LCtrl, break-NumLock sequence F0,14,F0,77 (p ⊖ p w)

This is an ASCII table annotated to show the keyboard keys that are represented by each character in my encoding. White-on-black cells are keys on map 2.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL ∅	SOH ⌜	STX ⊥	ETX ⌟	EOT ↯	ENQ ?	ACK ☑	BEL ☎	BS ⌫	HT ↦	LF ↓	VT ↧	FF ⇟	CR ↵	SO ⊙	SI ⊗
	Ext Shift	**Home**	**Insert**	**End**	**F4**	**F1**	**F3**	**F6**	**Bksp**	**Tab**	**KP Enter**	**Page Up**	**Page Down**	**Enter**	**Left Shift**	**Right Shift**
1	DLE ⌥	DC1 ◎	DC2 ⊛	DC3 ⊖	DC4 ⊜	NAK ☒	SYN ⏱	ETB ▣	CAN ✖	EM ■	SUB ⍰	ESC ⇑	FS ⇞	GS ⇟	RS ∮	US ∫
	PrtSc SysRq	**F5**	**Caps Lock**	**F8**	**Pause Break**	**F10**	**Scroll Lock**	**F12**	**F7**			**Esc**	**F11**	**F2**		**F9**
2	SP ␣	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
	Spc	**! 1**		**# 3**	**$ 4**	**% 5**	**& 7**	**“ ’**	**(9**	**) 0**	*** 8**	**KP +**	**< ,**	**KP -**	**KP .**	**KP /**
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	KP 0	**KP 1**	**KP 2**	**KP 3**	**KP 4**	**KP 5**	**KP 6**	**KP 7**	**KP 8**	**KP 9**		**: ;**		**+ =**	**> .**	**? /**
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	@ 2	**A a**	**B b**	**C c**	**D d**	**E e**	**F f**	**G g**	**H h**	**I i**	**J j**	**K k**	**L l**	**M m**	**N n**	**O o**
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	P p	**Q q**	**R r**	**S s**	**T t**	**U u**	**V v**	**W w**	**X x**	**Y y**	**Z z**	**{ [**	**| **	**}]**	**^ 6**	**_ -**
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	~ `	**Left Alt**	**Right Alt**		**↓**								**←**	**Menu**	**Num Lock**	
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL ⌦
			→			**↑**	**Left GUI**	**Right GUI**	**KP ***			**Left Ctrl**		**Right Ctrl**		**Delete**

- OUT OF DATE - The currently unused ASCII characters in this mapping are the following:

■ (EM)	⍰ (SUB)	∮ (RS)	“ (double quote)
: (colon)	< (less than)	c	e
f	g	h	i
j	k	o	p
q	s	t	y
z	| (vertical bar)	~ (tilde)	

Simplified Logic for Key Decoder module

State bits and their meanings:

f0 – We have just read an F0 byte, which means that the current code sequence is a break code (rather than a make code). The next byte received will terminate the current code sequence, and we will output a code with the high bit turned on to indicate that it is a break code.

e1 – We have just read the first byte (E1) of a secondary extended sequence; this flag will turn off after the next normal (less than 80) byte is received, but it will prevent that byte from immediately causing any output. This flag is only needed to process the Pause/Break key.

e0 – We are presently in the midst of an "extended" key code sequence that started with either an E0 or E1 byte code.

State table: (here "-" means "don't care;" and the "? :" operator abbreviates if-then-else)

Inputs to State-Machine Logic											**Outputs of State-Machine Logic**									
Current State Bits			**Byte In**								**Next State Bits**			**Byte Out**						
e1	**e0**	**f0**	**b_7**	**b_6**	**b_5**	**b_4**	**b_3**	**b_2**	**b_1**	**b_0**	**e1**	**e0**	**f0**	**b_7**	**b_6**	**b_5**	**b_4**	**b_3**	**b_2**	**b_1**	**b_0**
0	e_0	f_0	0	$b_{6..0}$							0	0	0	f_0	e_0 ? map2:map1($b_{6..0}$)						
-	-	f_0	1	0	-	-	-	-	-	-	0	0	0	f_0	18_{16} *(represents* F7 *key)*						
1	-	-	0	-	-	-	-	-	-	-	0	1	0	none *(skips 14 in Pause/Brk)*							
e_1	e_0	-	1	1	-	1	-	-	-	-	e_1	e_0	1	none *(received F0 byte)*							
-	-	-	1	1	-	0	-	-	-	b_0	b_0	1	0	none *(received E0/E1 byte)*							

1st row: (**norm_byte**) Last byte of a make or break code for a normal map1 or map2 key when not in the 2ndary extended mode. Output the 7-bit ASCII value looked up from map1 (or map2 if the e0 flag is on), and turn on bit 7 if f0 (break code).

2nd row: (**83_byte**) Last byte of a make/break code for the exceptional map1 key (F7).

3rd row: (**skip_byte**) Encountered a byte code below 80_{16} while in the 2nd ary extended mode. Ignore this byte and go to the primary extended mode. This case handles the 14_{16} (left-control) codes that occur in the Pause/Break sequence.

4th row: (**F0_byte**) Encountered a byte starting with D or F. Assume it's F0 and set the f0 flag, indicating that this is a break code. Don't change the extended-mode bits e0 and e1. This is needed to correctly handle E0 and E1 sequences.

5th row: (**E_byte**) Encountered a byte starting with C or E. Assume it's E0 or E1 and set the e0 extended-mode bit. Also set the e1 flag to go into 2ndary extended mode if the LSB is 1, indicating E1.

<u>Logic for identifying which row of the above table we are on:</u>

low_byte = ~b7

norm_byte = ~e1 & low_byte (1st row)

83_byte = b7 & ~b6 (2nd row)

skip_byte = e1 & low_byte (3rd row)

hi_byte = b7 & b6

F0_byte = hi_byte & b4 (4th row)

E_byte = hi_byte & ~b4 (5^{th} row)

Logic for determining the next-state bits based on the above:

e0* = e1 | (F0_byte & e0) | E_byte

Either we were just in e1 mode, or we just received F0 and were already in E0 mode, or we just received an E0 or E1 byte.

e1* = hi_byte & (e1 | (E_byte & b0))

The byte just seen was a high one, and either we were already in E1 mode, or we just received an E1 byte.

f0* = F0_byte

The byte just seen was an F0.

Some Notes

For the above simplified design to work, map 2 needs to be modified to include assignments for the following codes:

- **Code 12** ("imaginary extended shift key"), which may or may not appear when pressing Prt Scr / SysRq, Insert, Delete, Home, End, Page Up, Page Down, or the arrow keys, but does not affect its meaning and can be ignored by the application. Suggested ASCII code: 00 (NUL), Null, a character to be ignored.
- **Code 77**, which occurs (twice) in the sequence generated by Pause/Break. The above design will output separate make and break codes when this key is pressed (and nothing when it is released). Suggested ASCII code: 13 (DC3), Device Control 3, a secondary stop.
- **Code 7C**, which occurs in the make and break sequences generated by PrtScr/SysRq. Suggested ASCII code: 10 (DLE), Data Link Escape, a special low-level escape character. This key may also generate press/release events involving Code 12, but these can be safely ignored.

- END OF DOCUMENT -