The Toy Assembler language once used at Rhodes in Computer Science 201

Executable Instructions

The set of machine code functions available is quite small. Those marked * set/unset the N and Z flags, those marked C set/unset the C flag, those marked v clear the V flag, those marked V set/unset the V flag. An informal summary of their semantics follows:

```
Mnemonic         Hex Decimal      Function

NOP              00h   0    No operation (break point in Fast Trace mode)
HLT              01h   1    Halt program execution

CLA              02h   2    CPU.A := 0
CMA       v   *  03h   3    CPU.A := NOT CPU.A
INC           *  04h   4    CPU.A := CPU.A + 1
DEC           *  05h   5    CPU.A := CPU.A - 1
SHL       v C *  06h   6    Shift CPU.A one bit left
SHR       v C *  07h   7    Shift CPU.A one bit right
ASR       v C *  08h   8    Shift CPU.A one bit right, propogate sign bit

CLX              09h   9    CPU.X := 0
TAX              0Ah  10    CPU.X := CPU.A
INX           *  0Bh  11    CPU.X := CPU.X + 1
DEX           *  0Ch  12    CPU.X := CPU.X - 1
CLC         C    0Dh  13    CPU.C := 0
CMC         C    0Eh  14    CPU.C := NOT CPU.C
CLV       V      0Fh  15    CPU.V := 0
PSH              10h  16    CPU.SP := CPU.SP - 1; Mem[CPU.SP] := CPU.A
POP       v   *  11h  17    CPU.A := Mem[CPU.SP]; CPU.SP := CPU.SP + 1
RET              12h  18    CPU.PC := Mem[CPU.SP]; CPU.SP := CPU.SP + 1
```

The following allow for simple I/O in various interpretations:

```
INI       v   *  13h  19    CPU.A := ReadInt()
INH       v   *  14h  20    CPU.A := ReadHex()
INB       v   *  15h  21    CPU.A := ReadBinary()
INA       v   *  16h  22    CPU.A := ReadChar()
OTI              17h  23    WriteInt(CPU.A)
OTC              18h  24    WriteCard(CPU.A)
OTH              19h  25    WriteHex(CPU.A)
OTB              1Ah  26    WriteBinary(CPU.A)
OTA              1Bh  27    WriteChar(CPU.A)
```

The above are all single-byte instructions. The following are all double-byte instructions. The first set load and store between the registers and memory:

```
LDA  B    v   *  1Ch  28    CPU.A := Mem[B]
LDX  B    v   *  1Dh  29    CPU.A := Mem[B+CPU.X]
LDI  B    v   *  1Eh  30    CPU.A := B
LSP  B           1Fh  31    CPU.SP := Mem[B]
LSI  B           20h  32    CPU.SP := B
STA  B           21h  33    Mem[B] := CPU.A
STX  B           22h  34    Mem[B+CPU.X] := CPU.A
```

The following perform arithmetic (two's complement)

```
ADD  B    V C *  23h  35    CPU.A := CPU.A + Mem[B]
ADX  B    V C *  24h  36    CPU.A := CPU.A + Mem[B+CPU.X]
ADI  B    V C *  25h  37    CPU.A := CPU.A + B
ADC  B    V C *  26h  38    CPU.A := CPU.A + Mem[B] + CPU.C
ACX  B    V C *  27h  39    CPU.A := CPU.A + Mem[B+CPU.X] + CPU.C
ACI  B    V C *  28h  40    CPU.A := CPU.A + B + CPU.C
SUB  B    V C *  29h  41    CPU.A := CPU.A - Mem[B]
SBX  B    V C *  2Ah  42    CPU.A := CPU.A - Mem[B+CPU.X]
SBI  B    V C *  2Bh  43    CPU.A := CPU.A - B
SBC  B    V C *  2Ch  44    CPU.A := CPU.A - Mem[B] - CPU.C
SCX  B    V C *  2Dh  45    CPU.A := CPU.A - Mem[B+CPU.X] - CPU.C
SCI  B    V C *  2Eh  46    CPU.A := CPU.A - B - CPU.C
```

The following perform comparisons without altering the accumulator (effectively they subtract the operand from the accumulator)

```
CMP  B    V C *  2Fh  47    CPU.N := CPU.A < Mem[B];       CPU.Z := CPU.A = Mem[B]
CPX  B    V C *  30h  48    CPU.N := CPU.A < Mem[B+CPU.X]; CPU.Z := CPU.A = Mem[B+CPU.X]
CPI  B    V C *  31h  49    CPU.N := CPU.A < B;            CPU.Z := CPU.A = B
```

The following do bit-wise logical operations:

```
ANA  B    v   *  32h  50    CPU.A := CPU.A AND Mem[B]
ANX  B    v   *  33h  51    CPU.A := CPU.A AND Mem[B+CPU.X]
ANI  B    v   *  34h  52    CPU.A := CPU.A AND B
ORA  B    v   *  35h  53    CPU.A := CPU.A OR Mem[B]
ORX  B    v   *  36h  54    CPU.A := CPU.A OR Mem[B+CPU.X]
ORI  B    v   *  37h  55    CPU.A := CPU.A OR B
```

The following allow change in flow dependent on the current values of the processor flags:

```
BRN  B          38h  56    CPU.PC := B
BZE  B          39h  57    IF Z THEN CPU.PC := B END
BNZ  B          3Ah  58    IF NOT Z THEN CPU.PC := B END
BNG  B          3Bh  59    IF N THEN CPU.PC := B END
BPZ  B          3Ch  60    IF NOT N THEN CPU.PC := B END
BCS  B          3Dh  61    IF CPU.C THEN CPU.PC := B END
BCC  B          3Eh  62    IF NOT CPU.C THEN CPU.PC := B END
BVS  B          3Fh  63    IF CPU.V THEN CPU.PC := B END
BVC  B          40h  64    IF NOT CPU.V THEN CPU.PC := B END
```

The following are best used in conjunction with a comparison instruction. For example, assuming that one has just executed a `CPI Y` instruction:

```
BLT  B          41h  65    IF CPU.A < Y THEN CPU.PC := B END
BGE  B          42h  66    IF CPU.A >= Y THEN CPU.PC := B END
BLE  B          43h  67    IF CPU.A <= Y THEN CPU.PC := B END
BGT  B          44h  68    IF CPU.A > Y THEN CPU.PC := B END
```

The above assume the comparison implies "two's complement" arithmetic. The following assume the comparison implies "unsigned" arithmetic.

```
JLT  B          45h  69    IF CPU.A < Y THEN CPU.PC := B END
JGE  B          46h  70    IF CPU.A >= Y THEN CPU.PC := B END
JLE  B          47h  71    IF CPU.A <= Y THEN CPU.PC := B END
JGT  B          48h  72    IF CPU.A > Y THEN CPU.PC := B END
```

The following is used to set up simple subroutine calls:

```
JSR  B          49h  73    CPU.SP := CPU.SP - 1; Mem[CPU.SP] := CPU.PC + 2; CPU.PC := B
```

Identifiers and literals

- Labels start with a letter and contain letters and digits thereafter: `PAT  B  TOTAL  R2D2`
- A decimal literal (0 … 255) may be written in an obvious form: `234  34  56`
- A hexadecimal literal may be written as a decimal digit followed by hex digits and a trailing H: `0FFH`
- A binary literal may be written as a string of up to eight 0's and 1's followed by a trailing % : `0111%  01010101%`
- A string literal is written between double quotes: `"Hello Pat"`
- A character literal is written between single quotes: `'s'`
- String and character literals may contain the usual escape sequences: `"Pat \t said \"Good luck\" all"`

Address fields

In executable instructions, an address field may consist of a single label, number, or character, or a simple expression with + and - signs between terms:

```
LDI  034H         ;  value given by hexadecimal literal
STA  TOTAL        ;  address denoted by single label
ADD  TOTAL + 1    ;  address denoted by a single label offset by 1
LDI  'S' + 1      ;  value given by a simple expression
```

* may be used to denote "the address of this byte" (for example `LDI  *`)

Directives

```
                BEG              ; start of program code
                END              ; end of program code
                DS   10          ; Reserve 10 bytes of storage, unlabelled
        TOTAL   DS   1           ; Reserve one byte of storage, labelled TOTAL
        LIST    DS   10          ; byte [] LIST = new byte[10];
                DC   'a'         ; preset 1 byte to '.' (ASCII 46), unlabelled
        AGE     DC   64          ; 1 byte initialized to value 64
        NAME    DC   "Mr Muggins" ; 10 bytes initialized to character values
                                 ; String literals may be used only in DC directives
```

Examples

```
          BEG            ; O9.ASM - count the bits in a number
                         ; P.D. Terry, 2009

          CLA            ;
          STA  BITS      ; bits = 0;
          INI            ; CPU.A = IO.readInt();
 LOOP                    ; do {
          SHR            ;   CPU.A = CPU.A / 2
          BCC  EVEN      ;   if (CPU.A % 2 != 0) {
          STA  TEMP      ;     temp = CPU.A;      // save it
          LDA  BITS
          INC
          STA  BITS      ;     bits++;
          LDA  TEMP      ;     CPU.A = temp;      // restore it
 EVEN     BNZ  LOOP      ; } while (CPU.A != 0);
          LDA  BITS      ;
          OTC            ; IO.writeCard(bits);
          HLT            ; System.exit(0);
 TEMP     DS   1         ; byte temp;
 BITS     DS   1         ; byte bits;
          END


          BEG            ; 21.ASM - Hello world!  N times
                         ; P.D. Terry, 2009

          LDI  PROMPT
          JSR  WRSTR     ; IO.writeString("How many more times must I greet you with Hello World! ");;
          INI
          STA  COUNT     ; count = IO.readInt();
 WHILE    BZE  EXIT      ; while (count != 0) {
          LDI  STR
          JSR  WRSTR     ;   IO.writeString("Hello world!");
          JSR  WRLN      ;   IO.writeLine();
          LDA  COUNT
          DEC
          STA  COUNT     ;   count-- ;
          BRN  WHILE     ; }
 EXIT     HLT            ; System.exit(0);

 PROMPT   DC   "How many more times must I greet you with "
 STR      DC   "Hello world! "
          DC   0
 COUNT    DS   1

 WRSTR                   ; Subroutine to print a nul-terminated string
                         ; To use, code JSR WRSTR with CPU.A
                         ;   set up to store the address of first byte
          TAX            ; CPU.X = CPU.A to point to start of string
 WLOOP    LDX  0         ; while (str[CPU.X] != 0) {
          BZE  WEXIT     ;
          OTA            ;   IO.writeChar(CPU.A);
          INX            ;   CPU.X++;
          BRN  WLOOP     ; }
 WEXIT    RET            ; return;

 WRLN     LDI  0DH       ; Subroutine to output CR/LF sequence
          OTA            ; IO.writeChar(CR);
          LDI  0AH       ;
          OTA            ; IO.writeChar(LF);
          RET            ; return;

          END
```

ASCII table

00	0	<NUL>	10	16	<DLE>	20	32	SP	30	48	0	40	64	@	50	80	P	60	96	'	70	112	p
01	1	<SOH>	11	17	<DC1>	21	33	!	31	49	1	41	65	A	51	81	Q	61	97	a	71	113	q
02	2	<STX>	12	18	<DC2>	22	34	"	32	50	2	42	66	B	52	82	R	62	98	b	72	114	r
03	3	<ETX>	13	19	<DC3>	23	35	#	33	51	3	43	67	C	53	83	S	63	99	c	73	115	s
04	4	<EOT>	14	20	<DC4>	24	36	$	34	52	4	44	68	D	54	84	T	64	100	d	74	116	t
05	5	<ENQ>	15	21	<NAK>	25	37	%	35	53	5	45	69	E	55	85	U	65	101	e	75	117	u
06	6	<ACK>	16	22	<SYN>	26	38	&	36	54	6	46	70	F	56	86	V	66	102	f	76	118	v
07	7	<BEL>	17	23	<ETB>	27	39	'	37	55	7	47	71	G	57	87	W	67	103	g	77	119	w
08	8	<BS>	18	24	<CAN>	28	40	(	38	56	8	48	72	H	58	88	X	68	104	h	78	120	x
09	9	<HT>	19	25	<EM>	29	41	)	39	57	9	49	73	I	59	89	Y	69	105	i	79	121	y
0A	10	<LF>	1A	26	<SUB>	2A	42	*	3A	58	:	4A	74	J	5A	90	Z	6A	106	j	7A	122	z
0B	11	<VT>	1B	27	<ESC>	2B	43	+	3B	59	;	4B	75	K	5B	91	[	6B	107	k	7B	123	{
0C	12	<FF>	1C	28	<FS>	2C	44	,	3C	60	<	4C	76	L	5C	92	\	6C	108	l	7C	124	|
0D	13	<CR>	1D	29	<GS>	2D	45	-	3D	61	=	4D	77	M	5D	93	]	6D	109	m	7D	125	}
0E	14	<SO>	1E	30	<RS>	2E	46	.	3E	62	>	4E	78	N	5E	94	↑	6E	110	n	7E	126	~
0F	15	<SI>	1F	31	<US>	2F	47	/	3F	63	?	4F	79	O	5F	95	_	6F	111	o	7F	127	<DEL>

80	128	Ç	90	144	É	A0	160	á	B0	176	▒	C0	192	└	D0	208	╨	E0	224	α	F0	240	≡
81	129	ü	91	145	æ	A1	161	í	B1	177	▓	C1	193	┴	D1	209	╤	E1	225	β	F1	241	±
82	130	é	92	146	Æ	A2	162	ó	B2	178	▓	C2	194	┬	D2	210	╥	E2	226	Γ	F2	242	≥
83	131	â	93	147	ô	A3	163	ú	B3	179	▓	C3	195	├	D3	211	╙	E3	227	π	F3	243	≤
84	132	ä	94	148	ö	A4	164	ñ	B4	180	┤	C4	196	─	D4	212	╘	E4	228	Σ	F4	244	⌠
85	133	à	95	149	ò	A5	165	Ñ	B5	181	╡	C5	197	┼	D5	213	╒	E5	229	σ	F5	245	⌡
86	134	å	96	150	û	A6	166	ª	B6	182	╢	C6	198	╞	D6	214	╓	E6	230	μ	F6	246	÷
87	135	ç	97	151	ù	A7	167	º	B7	183	╖	C7	199	╟	D7	215	╫	E7	231	τ	F7	247	≈
88	136	ê	98	152	ÿ	A8	168	¿	B8	184	╕	C8	200	╚	D8	216	╪	E8	232	Φ	F8	248	°
89	137	ë	99	153	Ö	A9	169	⌐	B9	185	╣	C9	201	╔	D9	217	┘	E9	233	Θ	F9	249	•
8A	138	è	9A	154	Ü	AA	170	¬	BA	186	║	CA	202	╩	DA	218	┌	EA	234	Ω	FA	250	
8B	139	ï	9B	155	↑	AB	171	½	BB	187	╗	CB	203	╦	DB	219	▉	EB	235	δ	FB	251	√
8C	140	î	9C	156	£	AC	172	¼	BC	188	╝	CC	204	╠	DC	220	▄	EC	236	∞	FC	252	η
8D	141	ì	9D	157	↓	AD	173	¡	BD	189	╜	CD	205	═	DD	221	▌	ED	237	ϕ	FD	253	²
8E	142	Ä	9E	158	↕	AE	174	«	BE	190	╛	CE	206	╬	DE	222	▐	EE	238	ε	FE	254	▪
8F	143	Å	9F	159	↔	AF	175	»	BF	191	┐	CF	207	╧	DF	223	▀	EF	239	∩	FF	255	