LED DRIVER INSTRUCTIONS

INSTRUCTION SYNTAX	FUNCTION	EXAMPLE	16-BIT ASSEMBLED BIT SEQUENCE	ASSEMBLED CODE HEX
ramp time, PWM Time is a positive constant (0 to 0.484*PWM); PWM is a positive or negative constant (-255 to 255). Note: time is rounded by assembler if needed.	Output PWM with increasing / decreasing duty cycle.	**ramp** 0.6, 255 ;Ramp up to full scale over 0.6s	0000 1010 1111 1111	0A FF
		ramp 1.2,-255 ;Ramp down to zero over 1.2s	0001 0101 1111 1111	15 FF

ramp var1, prescale, var2
 	
Var1 is a variable (ra, rb, rc, rd);
Prescale is a boolean constant (pre=0 or pre=1);
Var2 is a variable (ra, rb, rc, rd).
	Output PWM with increasing / decreasing duty cycle.	ld ra, 31
		ld rb, 255
		ramp ra, pre=0,+rb
		
		;Ramp up to full scale over 3.9s
			1000 0100 0000 0001	84 01

		ld ra, 1
		ld rb, 255
		ramp ra, pre=0,-rb
		
		;Ramp down to zero over 0.12s
			1000 0100 0001 0001	84 11

set_pwm PWM PWM is a constant (0-255 or 0 - FFh).	Generate a continuous PWM output.	**set_pwm 128** ;Set PWM Duty-Cycle to 50%	0100 0000 1000 0000	4080
set_pwm var1 Var1 is a variable (ra, rb, rc, rd).	Generate a continuous PWM output.	ld rc, 128 **set_pwm rc** ;Set PWM Duty-Cycle to 50%	1000 0100 0110 0010	8462
wait time Time is a positive constant (0 to 0.484). Note: time is rounded by assembler if needed.	Pause for some time.	**wait 0.25** ;Wait 0.25 seconds	0110 0000 0000 0000	6000

LED MAPPING INSTRUCTIONS

INSTRUCTION SYNTAX	FUNCTION	EXAMPLE	16-BIT ASSEMBLED BIT SEQUENCE	ASSEMBLED CODE HEX
mux_ld_start address Address is a label which specifies where to find the first row.	Defines the start address of the mapping data table.	**mux_ld_start** row1 ; The first row can be found at the address marked with 'row1'	1001 1110 0000 0000 Assumed that "row1" points to addr 00h.	9E00
mux_map_start address Address is a label which specifies where to find the first row.	Defines the start address of the mapping data table and sets the row active.	**mux_map_start** row1 ; The first row can be found at the address marked with 'row1'	1001 1100 0000 0000 Assumed that "row1" points to addr 00h.	9C00
mux_ld_end address Address is a label which specifies where to find the last row.	Defines the end address of the mapping data table.	**mux_ld_end** row9 ; The last row can be found at the address marked with 'row9'	1001 1100 1000 1000 Assumed that "row9" points to addr 08h.	9C88
mux_sel output Output is a constant (0 to 9 or 16).	Connects one and only one LED output to an engine.	**mux_sel** 1 ; D1 output will be connected to the engine.	1001 1101 0000 0001	9D01
mux_clr	Clears engine-to-driver mapping.	**mux_clr**	1001 1101 0000 0000	9D00
mux_map_next	Sets the next row active in the mapping table.	**mux_map_next**	1001 1101 1000 0000	9D80
mux_map_prev	Sets the previous row active in the mapping table.	**mux_map_prev**	1001 1101 1100 0000	9DC0
mux_ld_next	The index pointer will be set to point to the next row in the mapping table.	**mux_ld_next**	1001 1101 1000 0001	9D81
mux_ld_prev	The index pointer will be set to point to the previous row in the mapping table.	**mux_ld_prev**	1001 1101 1100 0001	9DC1
mux_ld_addr address Address is a label which specifies the row to which the pointer is to be moved.	Sets the index pointer to point the mapping table row defined by address.	**mux_ld_addr** row2 ; The index pointer will be set to point to the row labelled with "row2".	1001 1111 0000 0001 Assumed that "row2" points to addr 01h.	9F01
mux_map_addr address Address is a label which specifies the row of the table that will be set active.	Sets the index pointer to point the mapping table row defined by address and sets the row active.	**mux_map_addr** row2 ; The index pointer will be set to point to the row labelled with "row2" and the row will be set active.	1001 1111 1000 0001 Assumed that "row2" points to addr 01h.	9F81

BRANCH INSTRUCTIONS

INSTRUCTION SYNTAX	FUNCTION	EXAMPLE	16-BIT ASSEMBLED BIT SEQUENCE	ASSEMBLED CODE HEX
`rst`	Resets program counter and start the program again.	`rst`	0000 0000 0000 0000	0000
branch `loopcount, address` `Loopcount` is a constant (0 to 63); `Address` is a label which specifies the offset.	Repeat a section of code.	**branch** `20, loop1` `; define loop for 20 times`	1010 1010 0000 0000 Assumed that "loop1" points to addr 00h.	AA00

branch `var1, address`
 	
`Var1` is a variable (ra, rb, rc, rd);
`Address` is a label which specifies the offset.
	Repeat a section of code.	`ld ra, 20`
		branch ra, `loop1`
		
		`; define loop for 20 times`
			1000 0110 0000 0000
			
			Assumed that "loop1" points to addr 00h.
				8600

int	Causes an interrupt.	**int**	1100 0100 0000 0000	C400
end `interrupt, reset` `Interrupt` (i) is an optional flag. `Reset` (r) is an optional flag.	End program execution.	**end i** `; End program execution and send an interrupt.`	1101 0000 0000 0000	D000
trigger `w{source1|source2...}` `Source` is the source of the trigger (1, 2, 3, e).	Wait a trigger.	**trigger** `w{1}` `;Wait a trigger from the engine 1.`	1110 0000 1000 0000	E080
trigger `s{target1|target2...}` `Target` is the target of the trigger (1, 2, 3, e).	Send a trigger.	**trigger** `s{1}` `;Send a trigger to the engine 1.`	1110 0000 0000 0010	E002
jne `var1, var2, address` `Var1` is a variable (ra, rb, rc, rd); `Var2` is a variable (ra, rb, rc, rd); `Address` is a label which specifies the offset.	Jump if not equal.	**jne** `ra, rb, flash` `;Jump to 'flash' if A != B.`	1000 1000 0010 0001 Assumed that offset = 2.	8821
jl `var1, var2, address` `Var1` is a variable (ra, rb, rc, rd); `Var2` is a variable (ra, rb, rc, rd); `Address` is a label which specifies the offset.	Jump if less.	**jl** `ra, rb, flash` `;Jump to 'flash' if A < B.`	1000 1010 0001 0001 **Assumed that offset = 1**	8A11

INSTRUCTION SYNTAX	FUNCTION	EXAMPLE	16-BIT ASSEMBLED BIT SEQUENCE	ASSEMBLED CODE HEX
jge var1, var2, address Var1 is a variable (ra, rb, rc, rd); Var2 is a variable (ra, rb, rc, rd); Address is a label which specifies the offset.	Jump if greater or equal.	**jge** ra, rb, flash ;Jump to 'flash' if A >= B.	1000 1100 0001 0001 Assumed that offset = 1.	8C11
je var1, var2, address Var1 is a variable (ra, rb, rc, rd); Var2 is a variable (ra, rb, rc, rd); Address is a label which specifies the offset.	Jump if equal.	**je** ra, rb, flash ;Jump to 'flash' if A = B.	1000 1110 0001 0001 Assumed that offset = 1.	8E11

DATA TRANSFER AND ARITHMETIC INSTRUCTIONS

INSTRUCTION SYNTAX	FUNCTION	EXAMPLE	16-BIT ASSEMBLED BIT SEQUENCE	ASSEMBLED CODE HEX
ld var, value Var is a variable (ra, rb, rc); Value is a constant (0 to 255 or 0 to FFh).	Assigns a value to a variable.	**ld** ra, 10 ;Variable A = 10.	1001 0000 0000 1010	900A
add var, value Var is a variable (ra, rb, rc); Value is a constant (0 to 255 or 0 to FFh).	Add the 8-bit value to the variable value.	**add** ra, 30 ;A = A + 30.	1001 0001 0001 1110	911E
add var1, var2, var3 Var1 is a variable (ra, rb, rc); Var2 is a variable (ra, rb, rc, rd); Var3 is a variable (ra, rb, rc, rd);	Add the value of var3 to the value of var2 and store the result in var1.	**add** ra, rc, rd ;A = C + D.	1001 0011 0000 1011	930B
sub var, value Var is a variable (ra, rb, rc); Value is a constant (0 to 255 or 0 to FFh).	Subtract the 8-bit value from the variable value.	**sub** ra, 30 ;A = A - 30.	1001 0010 0001 1110	921E
sub var1, var2, var3 Var1 is a variable (ra, rb, rc); Var2 is a variable (ra, rb, rc, rd); Var3 is a variable (ra, rb, rc, rd);	Subtract the value of var3 from the value of var2 and store the result in var1.	**sub** ra, rc, rd ;A = C - D	1001 0011 0001 1011	931B