

Using the Mic-1 Animation Library

All of the following take a **flag** parameter which may have one of the following values:

ANI_DISPLAY_NONE: do not display anything

ANI_DISPLAY_FAST: display the result but do not show any motion

ANI_DISPLAY_SLOW: normal display showing motion

```
void ani_display_all_memory(int flag);  
    redisplay all memory locations known to the display routines
```

```
void ani_initialize_sound_setup(char *s);  
    called by soundinit to set sound parameters from the file s.  
    The file contains a list of sound commands to set up the notes to play when certain actions occur.
```

```
void ani_move_from_mbr_to_memory(int loc, int value, int flag);  
    Move the value value from the mbr into the memory array at address loc
```

```
void ani_move_from_memory_to_mbr(int loc, int value, int flag);  
    Move value value from the memory array at location loc to the mbr
```

```
void ani_move_value_from_alatch_to_amux(int value, int flag);  
    Move value value from the alatch to the amux
```

```
void ani_move_value_from_alu_to_shifter(int value1, int value2, int flag);  
    Move the value value1 from the alu into the shifter with new value value2
```

```
void ani_move_value_from_blatch_to_mar(int value, int flag);  
    Move the value value from the blatch to the mar
```

```
void ani_move_value_from_inc_to_mpc(int value, int flag);  
    Move the value value from the incrementer to the mpc
```

```
void ani_move_value_from_mbr_to_amux(int value, int flag);  
    Move the value value from the mbr to the amux
```

```
void ani_move_value_from_mbr_to_memory(int value,int flag);  
    Move the value value from the mbr to the memory dot
```

```
void ani_move_value_from_memory_to_mbr(int value, int flag);  
    Move value value from memory dot to mbr
```

```
void ani_move_value_from_mir_addr_to_MPC(int value, int flag);  
    Move value value from mir address field to the MPC
```

```
void ani_move_value_from_mpc_to_inc(int value, int flag);  
    Move value value from the mpc to the incrementer and set the displayed value to value+1
```

```
void ani_move_value_from_reg_to_alatch(int value, int reg, int flag);  
    Move value value from register reg to alatch
```

```
void ani_move_value_from_reg_to_blatch(int value, int reg, int flag);  
    Move value value from register reg to blatch
```

```
void ani_move_value_from_shifter_to_mbr(int value, int flag);  
    Move value value from shifter to mbr
```

```
void ani_move_value_from_shifter_to_reg(int value, int reg, int flag);  
    Move value value from the shifter to register reg
```

```
void ani_movie_off(int flag);  
    Stop saving movie information by sending the movieoff command to the animator
```

```

void ani_movie_on(char *ename, double fps, int sf, int num, int type, int flag);
    Start saving movie information by sending the movieon command to the animator. The command sent is:
    movieon ename fps sf num type

void ani_remove_memory(int loc, int flag);
    Remove the memory entry at location loc from the memory display and redisplay memory

void ani_set_alatch_value(int val, int flag);
    Set the value displayed in the alatch to val

void ani_set_alu_nz(char nch, char zch, int flag);
    Set the value displayed in the blatch value to val

void ani_set_alu_value(int value, int flag);
    Set the value displayed above the ALU to value

void ani_set_a_memory_value(int loc,int value,int flag);
    Insert he value value into the memory array at location loc, delete a previous entry if there was no rrom for a new one.
    if flag is not ANI_DISPLAY_NONE, redisplay the memory

void ani_set_blatch_value(int val, int flag);
    Set the value displayed in the blatch to value

void ani_set_cycle_count_value(int value, int flag);
    Set the value displayed in the cycle count to value

void ani_set_decoded_ir(int val, int flag);
    Disassemble the MAC instruction val and display in the Disassembled IR box

void ani_set_decoded_mir(int val, int flag);
    Disassemble the MIC instruction val and display in the Disassembled MIR box

void ani_set_instruction_count_value(int value, int flag);
    Set the value shown in the Instruction Count box to value

void ani_set_mar_value(int val, int flag);
    Set the value shown in the MAR box to value

void ani_set_mbr_value(int val, int flag);
    Set the value shown in the MBR to value

void ani_set_memory_as_instruction(int loc, int value);
    If the location loc is in the memory array, set an internal flag indicating that it is an instruction and show its disassembled
    operation in the memory array

void ani_set_memory_pc(int loc, int flag);
    Set the PC arrow to point to memory location loc

void ani_set_memory_sp(int loc, int flag);
    Set the SP arrow to point to the memory location loc

void ani_set_mir_addr_value(int value, int flag);
    Set the value stored in the MIR address field to value (between 0 and 255)

void ani_set_mir_alu_value(char ch, int flag);
    Set the character displayed in the MIR ALU field to ch

void ani_set_mir_amux_value(char *str, int flag);
    Set the string displayed in the MIR AMUX field to str

void ani_set_mir_A_value(int val, int flag);
    Set the value displayed in the MIR A field to value (between 0 and 15)

```

```

void ani_set_mir_B_value(int val, int flag);
    Set the value displayed in the MIR B field to value (between 0 and 15)
void ani_set_mir_cond_value(char ch, int flag);
    Set the character displayed in the MIR COND field to ch
void ani_set_mir_C_value(int val, int flag);
    Set the value displayed in the MIR C field to value (between 0 and 15)
void ani_set_mir_enc_value(char ch, int flag);
    Set the character displayed in the MIR ENC field to ch
void ani_set_mir_mar_value(char ch, int flag);
    Set the character displayed in the MIR MAR field to ch
void ani_set_mir_mbr_value(char ch, int flag);
    Set the character displayed in the MIR MBR field to ch
void ani_set_mir_rd_value(char ch, int flag);
    Set the character displayed in the MIR RD field to ch
void ani_set_mir_sh_value(char ch, int flag);
    Set the character displayed in the MIR SH field to ch
void ani_set_mir_wr_value(char ch, int flag);
    Set the character displayed in the MIR WR field to ch
void ani_set_mpc_inc_value(int val, int flag);
    Set the value displayed in the MPC Incrementer to val
void ani_set_mpc_value(int val, int flag);
    Set the value displayed in the MPC to val
void ani_set_programmer_name(char *str, int flag);
    Set the programmer name displayed in the lower right corner to str
void ani_set_register_value(int value, int reg, int flag);
    Set the value displayed in register reg to value value
void ani_set_shifter_value(int value, int flag);
    Set the value displayed in the shifter to value
void ani_set_subcycle_active(int value, int flag);
    Set the subcycle active to value (value is between 1 and 4)
void ani_sound_volume(double volfact);
    Set the sound volume to volfact (between 0 and 1)
void decode_MAC_register(unsigned long op, unsigned long opcode_mask,
    unsigned long operand_mask, int type, char *name);
    Set the method for disassembling a MAC instruction.
    op contains the opcode and opcode mask masks the opcode.
    operand_mask masks the operand.
    type gives the type of instruction (Only INSTR_TYPE_16bit is supported)
    name gives the string to be displayed as the opcode portion of the instruction
void initialize_animator();
    Initialize the animator display.
void initialize_default_MAC();
    Initialize the data structure used for disassembling MAC instructions to the default
void initialize_MAC();

```

Clear the data structure used for disassembling MAC instructions

About Memory

The animator display keeps track of a memory array so that it can be efficiently redisplayed. For each memory value it keeps track of the address, the value, and a flag indicating whether it was an instruction. At most **MAXMEMDISP** values can be displayed, currently 30. Memory values are kept in order sorted by address. Initially no memory is displayed.

If you will not be using more than 30 memory locations, there is nothing you need to do that is special. After you read in your memory values, call **ani_set_a_memory_value** for each one using flag **ANI_DISPLAY_NONE** and then call **ani_display_all_memory**. This is much faster than calling **ani_set_a_memory_value** for each value with the flag set to display.

If there are memory locations that you do not want to display, use **ani_move_value_from_mbr_to_memory** instead of **ani_move_from_mbr_to_memory**.

The following routines allow memory values to be modified:

ani_set_memory_as_instruction

used to set a memory instruction flag. The location must have been previously set.

ani_display_all_memory

used to efficiently redisplay the memory known to the animator display if flags is not **ANI_DISPLAY_NONE**

ani_move_from_mbr_to_memory

If the memory location is already known to the animator, the value is set. Otherwise, a value is inserted in the memory array at the appropriate location and then the move takes place. When the memory array fills up no additional values are put in and no move is shown. If this represents a new memory location, all memory is efficiently redisplayed. If it corresponds to a previously known location, just that one is updated.

ani_set_a_memory_value

Sets an entry in the memory array if possible. Efficiently redisplay if flags is not **ANI_DISPLAY_NONE**