MCPC (MineCraft Personal Computer) - Architecture and hardware documentation

Stefan Reiter

2019-01-19

1 Introduction

This document provides a rough documentation of the MCPC (MineCraft Personal Computer) architecture and hardware. It does not go into detail on the inner workings of the different subsystems, but rather strives to provide a high-level description of components to allow developers to treat them as black-box systems in higher-abstraction systems.

2 Formatting

This document uses Verilog-style number formatting. See examples below:

- *h1C* refers to “1C” in hexadecimal notation
- *b1001* refers to “1001” in binary notation
- *42* refers to “42” in decimal notation (implicit base is 10)
- *h8b2* refers to the third bit of 8_{16}
- *16’h9* refers to a 16-bit number containing 0009_{16}

For memory locations, the following notation is used:

- [h1234] refers to the memory cell at 25’h0001234 (memory addressing is 25 bit total)
- [pA.h1234] refers to the memory cell at 25’h0015234 (“pA” denoting page 10_{10})

3 Assembly language instruction set

Memory locations

- EEPROM (read-only)
- SRAM
- General purpose registers (A, B, C, D, E, F, G, H) = h0-h7 (NOTE: Content of register H is displayed on a simple hex 7-segment)
- Special registers - SCR1 (Scratch) = h8, SCR2 (Scratch) = h9, SP (Stack Pointer) = hA, PC (Program Counter) = hB, 0 = hC, 1 = hD, -1 = hE, BUS (Last Received Bus Data, Deprecated) = hF

Misc. Details

- 16 bit words, no byte addressing
- Architecture is Little Endian: LSB is rightmost
- Scratch registers: General purpose, but not guaranteed to be consistent after library calls (but they are consistent if only base commands are used)

3.1 Instructions

Data instructions

- (h0) HALT
- (h1) MOV from.reg to.reg - moves data between registers
- (h2) MOVNZ from.reg to.reg if.reg - performs a MOV if the value in if.reg is not 0 (contains at least one set bit)
- (h3) MOVEZ from.reg to.reg if.reg - performs a MOV if the value in if.reg is 0 (contains no set bits)
- (h4) BUS data.reg addr.lit3 - sends data from register data.reg to the bus address specified in addr.lit3 (deprecated)
- (h5) MEMR [see extended information below]
- (h6) SET <ignored*> addr.reg - writes the following instruction directly to the register in addr (and skips it, thus not executing the following instruction) - Setting PC is NOT possible!
- (h7) MEMW [see extended information below]

*ignored means 4 bits that have no effect but padding; these are not present in assembler form

ALU instructions

- Format: INS op1.reg out.reg op2.reg
- INS: AND, OR, XOR, ADD, SHFT*, MUL, GT, EQ (h8-hF)

*A note on SHFT: The input value is shifted by the operator input masked via 16’h00FF, the other bits (Mask: 16’hFF00) are used to determine shifting direction. A left shift is performed iff any bit in the direction-masked input is 1, a right shift otherwise. All shifts are logical, not numerical.

3.2 Memory access and layout

MEMR instruction reference

```
.... xxxx xxxx 0101 (16 bit instruction)
|    |    |    |
|    |    |    + MEMR instruction code h5
|    |    |
|    |    + Register containing memory address
|    |
|    + Register to write read result to
|
+ unused
```

MEMW instruction reference

```
     xxxx .... xxxx 0111 (16 bit instruction)
     |    |    |    |
     |    |    |    + MEMW instruction code h7
     |    |    |
     |    |    + Register containing memory address
     |    |
     |    + unused
     |
     + Register containing data to write
```

Memory address definition

- [hF] … CFG bit (if set, action will be performed on CFG table entry)
- [h0-hE] … Access address

3.3 Fibonacci calculator

(example assembler code)

```
; Setup variables
MOV 1 A
MOV 1 B
MOV 1 D
; Set jump address
SET E
0x0007
; Set number of fibonacci numbers to calculate
SET F
0x0020
; Perform fibonacci algorithm
ADD A B C
MOV B A
MOV C B
; Store output in consecutive RAM addresses
STOR A D
INC D
; Loop
JMPGT E F D
; End
```

4 CFG register layout

4.1 General/Memory control

- [h8000] r … MCPC version number
- [h8001] rw(k) … RAM instruction loading (RIL; loads instruction from RAM instead of EEPROM)
- [h8002] rw(k) … RIL PC swap (program counter value to load on enabling/disabling RIL)
- [h8003] rw(k) … Protected memory page (Instructions executed as RIL when execute-select page matches this are executed with kernel priviledges, defaults to 0)
- [h8004] rw(k) … Kernel memory bits (see [h8800] below)
- [h8800] rw … Memory high bits (RAM uses 25 bit addressing, lower 15 bit are set by MEMR/MEMW directly, the next 5 bits are loaded from the lower bits of this CFG register, the missing 5 bits are the lower kernel memory bits)

4.2 Bootloader ROM

- [hD000-hD800] r … Bootloader ROM read-only access (up to h800 relative, e.g. 2048 words)

4.3 VGA subsystem

- [hDFFD] r … Width of display (maximum x, exclusive, 98 on hardware, 120 max)
- [hDFFE] r … Height of display (maximum y, exclusive, 35 on hardware, 65 max)
- [hDFFF] r … Last valid address of ASCII buffer
- [hE000-{hDFFF}] rw … ASCII buffer (one ascii characted per data word, high bits unused)

4.4 Interrupt/Timer subsystem

- [h9000] rw(k) … Interrupt handler address
- [h9001] rw(k) … Interrupt enable
- [h9002] rw(k) … Interrupt context state (can only write 0 to exit IRQ, other writes are ignored)
- [h9010-h9011] r(k) … Current interrupt data (reads 0 if [h9002] reads 0)

4.5 Virtualization subsystem

TBD

4.6 Debugging

- [hFFFF] w … Break Execution (enable manual clock)

5 FPGA

5.1 Physical control layout

Status output:

- 7-segment hex display:
  - SW9: Register H (SW5 up: from IRQ regs)
  - SW8: PC
  - SW7: Instruction buffer
  - SW6: Memory FIFO data read
  - no switch up: Debug information
- Status LEDs:
  - LEDR0: Halted
  - LEDR1: In IRQ handler/context
  - LEDR2: continue_execution bit
  - LEDR3: Debugger enabled
  - LEDR4: CLK
  - LEDR5: IF_EN
  - LEDR6: STATE: INS_LOAD
  - LEDR7: STATE: WAITING
  - LEDR8: STATE: COMMIT
  - LEDR9: STATE: PC_INC
- SW0:
  - Up: Auto clock
  - Down: Manual clock
- SW1:
  - On change: Trigger IRQ for keycode "A"

5.2 On-Chip hardware debugger

UART (115200 baud, n/8/1 mode)
Pins: ARDUINO[0] = RX, ARDUINO[1] = TX

Command syntax

- 8 bit
- bits 7-4: data
- bits 3-0: OP-code

OP-codes

Name	Address	Description
DEBUGGER_OPCODE_GET	h1	Set read address (3 bit) and print contents of specified register
DEBUGGER_OPCODE_SET	h2	Set write address (3 bit)
DEBUGGER_OPCODE_HI	h4	Write data to high bits at write address
DEBUGGER_OPCODE_LO	h8	Write data to low bits at write address
DEBUGGER_OPCODE_STEP	hC	Execute a single processor instruction and print hFF when done
DEBUGGER_OPCODE_DUMP_ROM	hE	Dumps the entire bootloader ROM to serial
DEBUGGER_OPCODE_DUMP_REGS	hA	Dumps all 16 registers to serial

Debug-Registers

Writeable:

- 0b0: Debug enable (dbgClk enable)
- 0b1: CPU Register address overwrite enable
- 0b2: Reset request
- 0b3: Instruction overwride enable
- 1b3-0: CPU Register address overwrite value
- 4b7-0: Low-bits of instruction overwrite buffer (first word)
- 5b7-0: High-bits of instruction overwrite buffer (first word)
- 6b7-0: Low-bits of instruction overwrite buffer (second word)
- 7b7-0: High-bits of instruction overwrite buffer (second word)

Readable:

- 8b7-0: Low-bit content of currently selected CPU register
- 9b7-0: High-bit content of currently selected CPU register
- Fb0: Halted

Known Issues

A program *starting* with a “SET” instruction can sometimes (unconfirmed reproducability) lead to issues in debugging mode.

6 Interrupts

Interrupts are added to a FIFO queue as two words:

- [0] … Interrupt number/code
- [1] … Data (arbitrary, useful in combination with [0])

To access these values from the interrupt handler, perform a read to the following CFG registers:

- [h9010] for [0]
- [h9011] for [1]

In CPU_STATE_PC_INC, the queue is checked, and if there is a valid entry it is removed and the interrupt handler (according to CFG[h9000]) is triggered. If CFG[h9001] (interrupt enable) is equal to 16'h0, the entry is removed, but the handler is not executed. The interrupt is thus discarded. If the FIFO queue is full (256 entries), further interrupts will be discarded.

To exit the interrupt handler, write 16'h0 to CFG[h9002] (interrupt context). CFG[h9002] will read 16'hFFFF iff the accessing MEMR command is executed in an interrupt handler context, 16'h0 otherwise.

To avoid breaking state for the interrupt program, all registers are temporarily stored in a backup environment. The interrupt contexts registers will be initialized to 0 (except 0, 1, -1 and PC).

TODO: Interrupts are always executed in kernel context.