MPROC Manual and Architecture Description

Christian Deussen

2013

Abstract

MPROC is a 16 bit CPU built out of TTL logic chips. The 74HCTxxx logic series is used.

Contents

1 Overview

1.1 Machine word

- Always one nibble of opcode bits and the second nibble includes which registers are involved.
- The next byte can be an immediate value.
- Data-bus is 8 bits wide.
- Address-bus is 15 bits wide: 0x0000 to 0x7FFF is the ram, 0x8000 to 0xFFFF is the flash!
- Thus we have 32kb flash and 32kb ram. We can execute both from ram and flash.
- The Pointer Register PTR is used for 16 memory access. See SET_PTR.
- PC is 16 bits wide: P_L:0...7; PC_H:8...15.
- ALU is 8 bits wide.

1.2 Stack

- 32kb Stack FIFO. The stack memory can only be addressed via PUSH and POP. No stack pointer arithmetic is possible because the stack is not connected to the address-bus. Thus we have a total of 32kb Ram +32KB Stack = 64kb RAM and 32kb ROM.
- The Stack pointer is implemented using 74xx193 binary counter chips.

1.3 I/O

- I/O can be read and written to with PUSH and POP.
- There are 2 output ports and one input port. All 8 bits wide.

1.4 Memory

- Ram(w24129ak12) and Flash(W29EE011) is connected to the same address Bus. The memory devices have both three-states outputs. The bank register(PTR) controls with the highest bit(bit14 on the address bus) whether RAM or the flash will be targeted. Writing to the flash will not work since a whole page(256bits) needs to be written at once. Maybe I can upgrade this later with a more decent flash chip, which will allow easy writing just like the ram.

1.5 Registers

Register	Used by	Note	Width
SP	PUSH/POP	Stack Pointer	16
PC	JMP[Z/C]	ProgramCounter	16
IR	-	Holds current instruction	16
LR	SAVE_LR, RET, PUSH/POP	Stores current return address	16
PTR	STR, LDR	Pointer Register	16
OutputReg[0..1]	POP	Output ports	8
InputReg0]	PUSH	Input ports	8
Reg[0..3]	MOV, STR, LDR, PUSH/POP	General Purpose registers	8

2 Instructions

2.1 JMP(C/Z)-instructions

- JMP[Z/C] number adds number to PC; [-128 < number < 128] This enables indirect Jumps.
- If the JMP has one argument, it is used as an offset instead as an address. Encoded with operand 0x00: reg1, number.
  For example JMP -17 jumps 17 bytes up.

2.2 Instruction Set

Nibble 2	Instruction	Note	Decoder
0x0	ADD regA, regB	regA + regB. Result in regA	0
0x1	SUB regA, regB	regA - regB. Result in regA	0
0x2	NOR regA, regB	regA = !(regA OR regB)	0
0x3	AND regA, regB	regA = regA AND regB	0
0x4	MOV regA, regB	regA = regB	0
0x5	MOVZ regA, regB	MOV if reg0 is zero	0
0x6	JMP regA, regB	set PC_H to regA, PC_L to regB	0
0x6	JMP number	Add sigend number to PC	0
0x7	JMPZ regA, regB	JMP if reg0 is zero	0
0x7	JMPZ number	Add sigend number to PC if reg0 is zero	0
0x8	JMPC regA, regB	JMP if carry is set	0
0x8	JMPC number	Add sigend number to PC if carry is set	0
0x9	STR regA	Store regA where PTR points to	2
0xA	LDR regA	Load into regA where PTR points to	1
0xB	STR_I regA	Store regA where PTR points to. Post increment PTR	2
0xC	LDR_I regA	Load into regA where PTR points to. Post increment PTR	1
0xD	SET_PTR regA, regB	Set PTR_H to regA, PTR_L to regB	0
0xE	PUSH regA	Push regA to the stack	2
0xF	POP regA	Pop stack item into regA	1
0xEF	RET	Restore LR in PC	
0xFF	SAVE_LR	Save PC + 1 in LR	

- regA is a register, regB is a register or a number.

2.3 Argument Decode Table 0

The second instruction Nibble defines the operand registers. Table 0 is used by MOV, and ALU
JMP(Z/C) uses "reg0, number" to encode "JMP(Z/C) PC_H, number" thus JMP(Z/C) reg0, number cant be used.

Nibble 1	Involved Registers
0x0	reg0, number
0x1	reg0, reg1
0x2	reg0, reg2
0x3	reg0, reg3
0x4	reg1, reg0
0x5	reg1, number
0x6	reg1, reg2
0x7	reg1, reg3
0x8	reg2, reg0
0x9	reg2, reg1
0xA	reg2, number
0xB	reg2, reg3
0xC	reg3, reg0
0xD	reg3, reg1
0xE	reg3, reg2
0xF	reg3, number

2.4 Argument Input Table 1

Used by LDR(_I)and POP

Nibble 1	Involved Registers
0x0	reg0
0x1	reg1
0x2	reg2
0x3	reg3
0x4	PTR_L
0x5	PTR_H
0x6	LR_LOW
0x7	LR_HIGH
0x8	PC_LOW
0x9	PC_HIGH
0xA	io_out0
0xB	io_out1
0xE	–used–
0xF	–used–

RET is encoded as POP with 0xE as argument nibble.
SAVE_LR is encoded as POP with 0F as argument nibble.

2.5 Argument Output Table 2

Used by STR(_I), PUSH

Nibble 1	Involved Registers
0x0	number
0x1	reg0
0x2	reg1
0x3	reg2
0x4	reg3
0x5	PTR_L
0x6	PTR_H
0x7	LR_L
0x8	LR_H
0x9	PC_L
0xA	PC_H
0xB	io_out0
0xC	io_out1
0xD	io_in0
0xE	io_in1

3 Calling Convention

- Caller saves his working registers(stack)
- Arguments are passed in Reg0 and Reg1. Additional arguments are passed via the stack.
- Return Address is saved in LR. If the callee wants to call other functions it has to save LR.
- Return values are in Reg0 and Reg1. Additional return values are on the stack.

4 Hardware Implementation

The 74HCT logic family is used. The design is not limited by the fanout. HCT chips have a typical fanout of >50. Propagation delay is 9ns, max frequency is 50MHz. PC and SP are implemented using 74xx193 binary counters.

4.1 Accessing Memory

Step	Read	Write
1	Apply address to A-bus, clear Not_OE and NOT_CS	Apply address to A-bus, Apply data to D-bus, clear Not_OE and NOT_CS
2	wait > 6ns	wait > 6ns
3	Read Data from Databus/Write read data back to registers	Set NOT_CS
4	Set NOT_CS	

4.2 Fetch and Decode

Step	Fetch and Decode
1	Connect PC to Mem Addr, IR to DBus
2	Do Mem Read and IR write Signals
3	Inrement PC. Twice when two byte command.

4.3 Execution Steps

Command	1	2	3	4
ALU	Write regA to DBUS. High to low triggered	Latch regA from DBUS for the ALU. Happens at the High to low transition.	Write RegB to DBUS, Latch ALU output at low to high edge of state-signal	Fill Reg
MOV(Z)	Write regB to DBUS	Fill Regs		
JMP(C/Z)	Write RegA to DBUS	Fill PC_LOW.	Write RegB to DBUS. This is done using the multiplexer to use an input selector as an ouput selector	Fill PC_HIGH
JMP(Offset)	Write PC_L to ALU	Latch PC_L to ALU input	Write RegB to DBUS, Fill PC_L with ALU and Inc or Dec PC_H depending on carry	
SET_PTR	Write RegB to DBUS	Fill PTR_L.	Write RegA to DBUS	Fill PTR_H.
SAVE_PTR				
RET				
LDR				
STR				
PUSH	Decrement SP			
POP			Increment SP	

5 Current Problems

5.1 DBUS Selector Encoding

SAVE_LR/RET need custom WRITE_XXX_DBUS signals. To be able to encode those for the dbus selector another multiplexer needs to be added.

5.2 SAVE_LR Implementation and RET

SAVE_LR does not know whether the following JMP instructions takes one or two bytes. How to decide whether to save PC+1 with SAVE_LR or PC+2?

5.3 ALU

ALU carry_out signal is changed by NOR and AND instructions. Need a flip flop or something similar. Do we need a flip flop for REG1_ZERO_MOVZ if reg1 is changed during a MOVZ instruction(Which could in turn have an effect on the STATE_SIGNAL of the MOV instruction)?

5.4 Startup

How to initialize the registers when power is turned on? RING_CNTR_CLR needs to go high. This clears the Ring Counter. Is this signal low active? CLR_ALL_REGS_NOT needs to go low for a short period and then for the rest of the time HIGH

5.5 Memory Access

Read: Can we apply the address, clear_not_oe and clear_not_cs in one instruction and fill the registers in the following instruction? 2 cycle memory access would be possible with this!

6 Toolchain

6.1 Assembler

Supports .define and CALL macro. CALL regA, regB equals to SAVE_LR; CALL regA, regB

6.2 Emulator

Does not yet support timing simulation. It just executes the binary