Shenzhen I/O ISA

Assembly Language

Instructions in the MCxxxxvassembly language are quite fluid-- they are often able to be written with either registers or immediates as inputs, and with the different flavors of registers (storage, simple I/O, and XBus) as outputs, without any additional specification from the user. Once this code is assembled into machine code, however, these specifications must be made explicit. For the user, however, this fluidity is a great asset.

Syntax

Most commands in the MCxxxx follow the format `command arg arg,` with a command followed by one or more arguments. These arguments can, more often than not, be either registers or specific numerical values (immediates).

Sleeping

The MC3999 chip operates on two clocks: a traditional CPU clock, and a real-world scale “long” clock. In game, the traditional CPU clock operates so fast as to be trivial for any interactions in the real world, while the long clock operates at a period of ~1 second. This long clock allows users to easily sync the operation of the MC3999 to real world I/O devices.

The `slp` command causes the chip to cease operation until a specified number of rising edges of the long clock have elapsed.

Registers

The MC3999 (the only chip we have implemented) has 5 registers available:

- `acc:` The internal accumulator register, all of the arithmetic operations store their results in this register. It is the only internal storage available on the MC3999.
- `p0, p1:` The two simple I/O ports. These ports can input and output values between 0 and 127. When output to, they will hold the outputted value until it is either overwritten, or if the corresponding port’s input value is read.
- `x0, x1:` The two XBus I/O ports. XBus ports, unlike simple I/O ports, are blocking. When a chip attempts to write to/read from one of these ports, it will stall until it is read from/written to. XBus inputs can handle values from -1024 to 1023.

Conditional Execution

The MC3999 currently has three commands that enable conditional execution: `teq, tgt`, and `tlt`. Each of these commands compares two values, and if the comparison is true, it stores the

chip's condition as "+". Else, it stores the chip's condition as "-" (initially, the chip has no internal condition).

Commands can be conditionally executed by preceding the command by a `+` or `-`. When a line with a conditional flag is reached, if that flag matches the chip's internal condition, that line will be executed. If it does not, then the line will not be executed (a `nop` will be executed in its place).

Jumping

The MC3999 supports jumping to predefined labels in programs. A label is a line of code of the format "LABEL:" (note the colon after the label). A `jmp LABEL` command will jump execution to the line after that predefined label. By default, programs jump back to the first line of code after the end of the program is reached.

Assembly Instructions

Instruction	Effect
nop	No Effect
mov R/I R	Saves/outputs the value stored in/specified by the first argument to the register specified by the second argument
jmp LABEL	Moves the program execution to the line specified by the label[1]
slp R/I	Sleeps the processor for a number of long clock cycles stored in/specified by the argument
teq R/I R/I	If the value stored in/specified by the first argument is equal to the value stored in/specified by the second argument, then set the internal condition to “+”. Else, set it to “-”
tgt R/I R/I	If the value stored in/specified by the first argument is larger than the value stored in/specified by the second argument, then set the internal condition to “+”. Else, set it to “-”
tlt R/I R/I	If the value stored in/specified by the first argument is less than the value stored in/specified by the second argument, then set the internal condition to “+”. Else, set it to “-”
add R/I	Add the value specified by/stored in the argument to acc, and store the result in acc
sub R/I	Subtract the value specified by/stored in the argument from acc, and store the result in acc
mul R/I	Multiply the value specified by/stored in the argument by acc, and store the result in acc
not	If the value stored in acc is 127, store 0 in acc. Else, store 127 in acc.

[1]*Labels are lines of the format “LABEL:”*

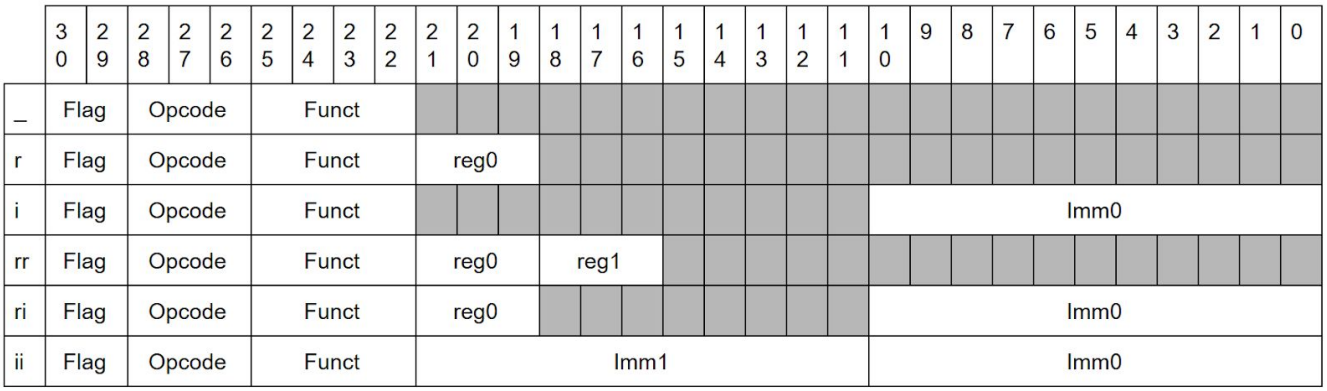
Machine Code

As these commands require a very diverse mix of possible input and output types, the machine code instructions have been structured to enable that flexibility. Each machine code instruction consists of the following components:

- Flag: A conditional execution flag, either +, -, or none. If a + or - flag is present, that instruction's execution will depend on the MC3999's internal conditional state when the instruction is reached in program memory.
- Opcode: This 3 bit code specifies the structure of the instruction's inputs/outputs, i.e. how many registers and immediates are present in the command.
- Funct: This 4 bit code specifies which instruction (e.g. `add`, `mov`) is being executed.

Additionally, most instructions also include register addresses (3 bits) and/or immediate values (11 bits). Instructions are 31 bits long, which is the length required of an instruction with two immediates (which, at this moment, is only possible with a comparison operation such as `teq`).

Instruction Structure



Assembly to Binary

Conditional Flags		Opcodes		Functs		Registers	
__	00	__	000	nop	0000	acc	000
+	01	r	010	mov	0001	dat*	001
-	10	i	001	jmp	0010	p0	010
		rr	110	slp	0011	p1	011
		ri	111	slx*	0100	x0	100
		ii	101	teq	1100	x1	101
				tgt	1101	x2*	110
				tlt	1110	x3*	111
				tcp*	1111		
				add	1000		
				sub	1001		
				mul	1010		
				not	1011		
				dgt*	0111		
				dst*	0110		

** : Not yet implemented*

RTL

Because machine code instructions require much more specificity than the assembly instructions, the number of possible machine code instructions is much higher than the assembly instructions. Machine code instructions are named based on their funct and opcode, in that order (e.g. `addi`, `movrr`). Their specific effects are listed in the following table:

Instruction	RTL
`nop`	N/A
`movrr`	R[reg1] = R[reg0]
`movri`	R[reg0] = Imm0
`jmpi`	PC = Imm0[1]
`slpr`	sleepVal[2] = R[reg0]
`slpi`	sleepVal = Imm0
`addr`	R[acc] = R[acc] + R[reg0]
`addi`	R[acc] = R[acc] + Imm0
`subr`	R[acc] = R[acc] - R[reg0]
`subi`	R[acc] = R[acc] - Imm0
`mulr`	R[acc] = R[acc] * R[reg0]
`muli`	R[acc] = R[acc] * Imm0
`not`	If R[acc] == 127, R[acc] = 0; else, R[acc] = 127
`teqrr`	If R[reg0] == R[reg1], cond[3] = +; else, cond = -
`teqri`	If R[reg0] == Imm0, cond = +; else, cond = -
`teqii`	If Imm0 == Imm1, cond = +; else, cond = -
`tgtrr`	If R[reg0] > R[reg1], cond = +; else, cond = -
`tgtri`	If R[reg0] > Imm0[4], cond = +; else, cond = -
`tgtii`	If Imm0 > Imm1, cond = +; else, cond = -
`tltrr`	If R[reg0] < R[reg1], cond = +; else, cond = -
`tltri`	If R[reg0] < Imm0[4], cond = +; else, cond = -
`tltii`	If Imm0 < Imm1, cond = +; else, cond = -

[1]This Imm0 is determined by the assembler based on the LABEL of the `jmp` command
[2]sleepVal is an internal variable that counts down the remaining long clock cycles until the MC3999 stops sleeping

[3] *cond is the internal variable for the MC3999's conditional execution state.*

[4] *`tgtri` and `tltri` have specific orders for the register and immediate, while the assembly commands `tgt` and `tlt` can have the register and immediate in any order. The assembler determines whether such an instruction compiles to `tgtri` or `tltri`. If, for example, an assembly command is `tgt I R`, it is assembled to a `tltri` instruction.*