The MiniMicro II CPU Architecture

Michael A. Gohde

November 17, 2015

1 Introduction

This document will provide an in-depth overview of the MiniMicro II CPU architecture. MiniMicro II is a big-endian, 16 bit, pipelined architecture with a 16 bit memory bus and 32 bit memory addressing. While standardized benchmarking is not yet possible at this time due to the lack of a working C compiler (a port is currently in progress), MiniMicro II should be able to execute, on average, 0.48 instructions per clock cycle (*see benchmarking section*).

2 Internal Design and Modeling

MiniMicro II is a soft CPU core implemented in the Verilog HDL. This allows it to be easily distributed, simulated, debugged, and possibly synthesized on FPGA or CPLD devices. The Verilog model itself is comprised of several behavioral components, while register transfer modeling was selected for pipeline management and more complex control instructions.

2.1 Registers

The core itself is internally completely 16 bit, with eight general purpose registers. All instructions excluding the Load Literal instruction are completely orthogonal for all registers in the set, however the core also features an eight bit flag register and 32-bit stack pointer, which can be addressed with some limitations through the use of certain instructions.

2.2 Memory Addressing Modes

MiniMicro II features four addressing modes with each dependent on the instruction executed.

2.2.1 ALU Instructions

All ALU instructions may operate only on the processor's internal general purpose registers.

2.2.2 Load and Store Instructions

MiniMicro II features six load and store instructions. The first two, aptly named LOAD and STORE (*see instruction table*) operate on the address formed when combining two registers (*see instruction format subsection*). The last four each operate on the stack register to either load or store a general purpose register value or to load or store the current value of the program counter.

2.2.3 Literals

There is one constant load instruction that may store a 15 bit constant into any of the general purpose registers. The 15 bit constant represents the lower 15 bits of a value, and the top bit is always fixed at zero.

2.3 The ALU

MiniMicro II features a 16 bit ALU capable of working with unsigned integer values (signed values should be computable with appropriate use of the flag register). The ALU lacks the ability to multiply or divide, however such capabilities will be introduced in future revisions of the MiniMicro soft core family.

2.4 Instruction Format

The MiniMicro architecture features two instruction formats.

2.4.1 Format 0

Instruction Format 0 is used by all instructions except for one. This format allows for the specification of one 6-bit operation code along with three 3-bit register operands. An instruction template is listed below:

F IIIIII AAA BBB CCC

From left to right (highest to lowest order), there exists the (F)ormat bit, six bits denoting the (I)nstruction, and the three regsister operands, A, B, and C. For all instructions, operands A and B denote the source data to compute, while operand C denotes the register to write to. As such, all instruction mnemonics should be followed explicitly with all register operands or, as noted in the next subsection, a constant value.

2.4.2 Format 1

Instruction Format 1 is used only by the Load Literal instruction. Its format specification doubles as its operation code in order to provide as much storage space utilization as possible. A Format 1 instruction template is listed below:

F DDDDDDDDDDDDDDD

From left to right (highest to lowest order), there exists the combined (F)ormat bit/operation code bit, as well as a 15-bit constant. This is one of the few non-orthogonal instructions, as the destination for all constant loads is register 7 (3'b111).

2.5 The Instruction Pipeline

MiniMicro II features an instruction pipeline consisting of four stages with provisions to allow instructions to execute over five stages if necessary. The four stages over which most instructions execute are, in order, Fetch, Data Dependency Resolve, Execute, and Writeback. Resolving data dependencies occupies an entire cycle as it is possible for chains of instructions with non-standard intermediate storage to execute immediately prior to any instruction that depends on them. All integer instructions as well as those that deal with data transfers and comparisons within the processor itself are pipelined, while all jumps, calls, or memory instructions execute in multiple cycles. These and other such factors are present in the instruction table below.

3 Basic Assembly Syntax

The MiniMicro II package currently includes a very basic assembler capable of generating all of the instructions listed in the Instruction Table section below. Because the assembler will include its own documentation, only basic syntax will be included.

3.1 Format 0 Instructions

All format 0 instructions consist of a single line containing a lowercase mnemonic and three register specifications. All registers are named with the lowercase letter 'r' followed by a number from 0 to 7. An example instruction is below:

```
add r7,r1,r2
```

The above instruction would add the contents of register 7 to register 1 and store the result in register 2. Please note that the following is also valid:

```
add r7,r7,r7
```

Where register 7 would be added and then stored back into itself.

3.2 Format 1 Instruction

The load literal instruction has a single string rather than a register specification. This string may be prefixed either with a hash sign '#' or a dollar sign '$', to represent the address at a label or a decimal numeric constant, respectively.

```
ldl $123
```

Would load the numeric value 123 into register 7.

```
ldl #mylabel
```

Would load the address of label 'mylabel' into register 7.

3.3 Labels

All labels represent a specific address that can be loaded as described above. Labels are prefixed with the '>' character.

```
>mylabel
ldl #mylabel
```

Would load the address of mylabel (incidentally the address of the LDL instruction) into register 7.

3.4 Constant Data

In order to include constant data in a program, it is necessary to prefix the line of data with a period.

```
>mylabel
. 1,2,3,4
```

The above would include four unsigned integer words with the values 1, 2, 3, and 4 in that order in the program binary. The label is placed as a means of easily accessing the data later.

4 Instruction Table

Mnemonic	OpCode (Dec)	OpCode (Hex)	"Cycles"	Repr.	Notes
NOP	0	0	1	n/a	Does nothing.
ADD	1	1	1	c=a+b	
ADDC	2	2	1	c=a+b	c=a+b+1 if carry bit set
SUB	3	3	1	c=a-b	
SUBC	4	4	1	c=a-b	c=a-b-1 if carry bit set
BSL	5	5	1	c=a<<b	
BSR	6	6	1	c=a>>b	
AND	7	7	1	c=a&b	

OR	8	8	1	c=a|b	
INV	9	9	1	c=˜a	b not used but must be present.
XOR	10	A	1	c=aˆb	
SSR	40	28	1	c=SR[15:0]	Writes lower 16 bits of stack register to c.
LSR	41	29	1	SR[31:16]=a SR[15:0]=b	Writes a and b to SR.
PUSH	42	2A	4	M[SR]=a	Pushes a
POP	43	2B	4	c=M[SR]	Pops c
CALL	44	2C	5	call A:B	Calls combined a and b.
RET	45	2D	5	jmp M[SR-1]:M[SR]	Read pop PC from stack.
CMP	50	32	1		Compare a and b through subtraction. Comparison flags are not affected by the ALU.
JMP	51	33	4		Jump to combined a and b.
JE	52	34	4		Jump if equal flag set.
JP	53	35	4		Jump if positive flag set.
JN	54	36	4		Jump is negative flag set.
HLT	59	3B	n/a		Halts processor execution.
LOAD	60	3C	4	c=M[a,b]	Loads from combined a, b.
STORE	61	3D	4	M[a,b]=c	Stores c to combined a, b.

LDFLGS	62	3E	1	Flags=a[7:0]	Loads processor flags from a.
STFLAGS	63	3F	1	c[7:0]=Flags	Store processor flags in c.
LDL	64+	40+	1		Loads lower 15 bits of instruction into r7.

5 Optimization

Overall, the optimization rules for MiniMicro II are fairly simple. They are:

1. Avoid branches if possible. Four ALU operations can be executed within one branch instruction.
2. Avoid tight loops. Due to the aforementioned branch penalty, loops should get as much work finished as possible in each iteration.
3. Avoid recursion. The CALL and RET instructions execute in five cycles, thus making loops far faster.
4. Allocate registers to constants. Constants like 1 and 0 should be kept for as long as possible, even though the LDL instruction is pipelined.

6 Benchmarking

For validation and testing, a number of programs were written for the MiniMicro architecture. One program in particular was made to demonstrate and test a wide variety of features including consecutive operations, data coherence, stack operations, conditional jumps, calls, and returns. This program, reproduced below, executed in 326 cycles in simulation and was used to calculate the 0.48 instructions per clock cycle figure listed above. The program is reproduced on the next page in its entirety.

```
ldl #main
jmp r6,r7,r0
>multfunc
xor r1,r1,r1
xor r2,r2,r2
ldl $1
and r7,r7,r3
>loop
add r6,r1,r1
add r3,r2,r2
ldl #out
cmp r2,r5,r0
je r0,r7,r0
ldl #loop
jmp r0,r7,r0
>out
ret r0,r0,r0
>main
ldl $10
and r7,r7,r6
and r7,r7,r5
ldl #multfunc
call r0,r7,r0
push r1,r0,r0
ldl #multfunc
call r0,r7,r0
pop r0,r0,r2
add r1,r2,r0
hlt
```