FUBAR RISC Computer Architecture

Roland Kammerer, Christian Paukovits, Mark Volcic, Gernot Vormayr

October 15, 2007

1 General

- 16 16-bit registers, r0 ... r15
- any register as return address
- little endian
- flags of the CPU:
  - Z: all bits of the last result are zero
  - C: "17^{th} bit" of the last result
  - N: 16^{th} bit of the last result
  - V: overflow, after sub/cmp it is $r1(15) \oplus r2(15) \oplus N \oplus C$, the latter two according to the result, other operations accordingly
  - I: allow interrupts
  - P: parity of the last result
- flags of the co-processor:
  - pr: prime number flag of the co-processor
  - fin: the co-processor has finished a calculation, and the result is available in register res

  Flags are written where meaningful: P ans Z are computed whenever a register is written, arithmetic operations may change C, N and V, interrupts clear I upon entry.
- flags are stored and restored upon interrupt entry and exit to/from "shflags" (shadow flags)
- 16 external interrupts, interrupt flag register
- support for hardware accelerated instructions for mathematical primitives by means of a co-processor
- support for power save modes

The processor uses a Harvard architecture; although it has not prevailed in mainstream-architectures, it is still used in embedded processors such as the Atmel AVR. The separation of code- and data-memory is not flexible enough for mainstream systems, but with small embedded processors the program code tends to be fixed anyway. A Harvard architecture enables the processor to make use of more memory , and the program code can be read from a ROM directly. A transient failure thus cannot destroy the program by overwriting its code section.

The data word for instruction as well as data memory is 16 bit wide. The number of data words for instruction memory is *not specified yet*, and for the data memory *not specified yet*.

2 Instruction Set

2.1 Instruction formats

The following formats for instructions are to be used:

Format	Bits 15 ... 12	Bits 11 ... 8	Bits 7 ... 4	Bits 3 ... 0
$\mathcal{A}$	Opcode	r3	r2	r1
$\mathcal{B}$	Opcode	n8		r1
$\mathcal{C}$	Opcode		r2	r1
$\mathcal{D}$	Opcode		n4	r1
$\mathcal{E}$	Opcode			r1
$\mathcal{F}$	Opcode			

2.2 Instruction Set of CPU

Format	Instruction	Opcode	Semantics
$\mathcal{A}$	add r1, r2, r3	0001	$r1 + r2 \rightarrow r3$
	sub r1, r2, r3	0010	$r1 - r2 \rightarrow r3$
	addc r1, r2, r3	0011	$r1 + r2 + C \rightarrow r3$
	subb r1, r2, r3	0100	$r1 - r2 - C \rightarrow r3$
	and r1, r2, r3	0101	$r1 \wedge r2 \rightarrow r3$
	or r1, r2, r3	0110	$r1 \vee r2 \rightarrow r3$
	xor r1, r2, r3	0111	$r1 \oplus r2 \rightarrow r3$
	mul r1, r2, r3	1000	$r1 * r2 \rightarrow r3$
	div r1, r2, r3	1001	$r1 \div r2 \rightarrow r3$
	udiv r1, r2, r3	1010	$r1 \div r2 \rightarrow r3$, unsigned
	mod r1, r2, r3	1011	$r1 \bmod r2 \rightarrow r3$
	umod r1, r2, r3	1100	$r1 \bmod r2 \rightarrow r3$, unsigned
$\mathcal{B}$	ldil r1, n8	1101	$n8 \rightarrow r1(7{:}0)$
	ldih r1, n8	1110	$n8 \rightarrow r1(15{:}8)$
$\mathcal{C}$	mov r1, r2	0000 0001	$r1 \rightarrow r2$
	not r1, r2	0000 0010	$\neg r1 \rightarrow r2$
	neg r1, r2	0000 0011	$-r1 \rightarrow r2$
	cmp r1, r2	0000 0100	$r1 - r2$, set flags
	shl r1, r2	0000 0101	$r1 \ll r2 \rightarrow r1$
	shr r1, r2	0000 0110	$r1 \ggg r2 \rightarrow r1$
	sar r1, r2	0000 0111	$r1 \gg r2 \rightarrow r1$
	rolc r1, r2	0000 1000	$(r1 \ll r2) \vee (C \ll (r2 - 1)) \vee (r1 \ggg (16 - r2 - 1))$
	rorc r1, r2	0000 1001	$(r1 \ggg r2) \vee (C \ll (16 - r2)) \vee (r1 \ll (16 - r2 - 1))$
	ld r1, r2	0000 1010	$[r1]{:}[r1 + 1] \rightarrow r2$
	st r1, r2	0000 1011	$r2 \rightarrow [r1]{:}[r1 + 1]$
$\mathcal{D}$	bset r1, n4	0000 1100	$r1 \vee (1 \ll n4) \rightarrow r1,\ 0 \leq n4 \leq 15$
	bclr r1, n4	0000 1101	$r1 \wedge \neg(1 \ll n4) \rightarrow r1,\ 0 \leq n4 \leq 15$
	btst r1, n4	0000 1110	$r1 \ggg n4 \wedge 1 \rightarrow Z,\ 0 \leq n4 \leq 15$

$\mathcal{E}$	jmp r1	0000 0000 0001	$r1 \rightarrow pc$
	jz r1	0000 0000 0010	$Z = 1 \Rightarrow r1 \rightarrow pc$
	jnz r1	0000 0000 0011	$Z = 0 \Rightarrow r1 \rightarrow pc$
	jle r1	0000 0000 0100	$\leq \ldots (Z = 1) \vee (N \not= V) \Rightarrow r1 \rightarrow pc$
	jlt r1	0000 0000 0101	$< \ldots (Z = 0) \wedge (N \not= V) \Rightarrow r1 \rightarrow pc$
	jge r1	0000 0000 0110	$\geq \ldots (Z = 1) \vee (N = V) \Rightarrow r1 \rightarrow pc$
	jgt r1	0000 0000 0111	$> \ldots (Z = 0) \wedge (N = V) \Rightarrow r1 \rightarrow pc$
	jule r1	0000 0000 1000	$\leq \ldots (Z = 1) \vee (C = 1) \Rightarrow r1 \rightarrow pc$
	jult r1	0000 0000 1001	$< \ldots (Z = 0) \wedge (C = 1) \Rightarrow r1 \rightarrow pc$
	juge r1	0000 0000 1010	$\geq \ldots (Z = 1) \vee (C = 0) \Rightarrow r1 \rightarrow pc$
	jugt r1	0000 0000 1011	$> \ldots (Z = 0) \wedge (C = 0) \Rightarrow r1 \rightarrow pc$
	push r1	0000 0000 1100	$r1 \rightarrow [sp],\ sp - -$
	pop r1	0000 0000 1101	$[sp] \rightarrow r1,\ sp + +$
	call r1	0000 0000 1110	$pc \rightarrow [sp],\ sp - -$
$\mathcal{F}$	nop	0000 0000 0000 0000	do nothing
	ret	0000 0000 0000 0001	$[sp] \rightarrow pc,\ sp + +$
	reti	0000 0000 0000 0010	$[sp] \rightarrow pc,\ sp + +,\ shflags \rightarrow flags$, clear interrupt flags
	sei	0000 0000 0000 0011	$1 \rightarrow I$
	cli	0000 0000 0000 0100	$0 \rightarrow I$
	eje	0000 0000 0000 1101	eject user from chair
	dtr	0000 0000 0000 1110	self destruct
	snz	0000 0000 0000 1111	snooze until interrupt or co-processor

2.2.1 unused opcodes of CPU

There is space left for 12 more opcodes:

2-register opcodes (1) : `00001111`

1-register opcodes (1) : `000000001111`

0-register opcodes (10) : from `0000000000000101` to `0000000000001110` (inclusively)

2.3 Instruction set of co-processor

The co-processor provides an additional 32-bit register res that holds the result of an operation. An additional flag pr indicates, whether a register content is prime in combination with the "prm" operation. Note that there are only explicit jump operations that consider the pr flag. Instead you have to conduct a bit test in the $flags$ register. Another flag is the fin flag that indicates a valid result in res.

Format	Instruction	Opcode	Semantics
$\mathcal{C}$	pow r1, r2	1111 0000	$r1^{r2} \rightarrow res$
	sum r1, r2	1111 0001	$\sum x \rightarrow res,\ x \in [r1] \ldots [r1 + r2]$
	prod r1, r2	1111 0010	$\prod x \rightarrow res,\ x \in [r1] \ldots [r1 + r2]$
	min r1, r2	1111 0011	$\min x \rightarrow res,\ x \in [r1] \ldots [r1 + r2]$
	max r1, r2	1111 0100	$\max x \rightarrow res,\ x \in [r1] \ldots [r1 + r2]$

	avg r1, r2	`1111 0101`	$\overline{[r1] + \ldots + [r1 + r2]} \rightarrow res$
$\mathcal{E}$	ldrlw r1	`1111 1111 0000`	$res(15:0) \rightarrow r1$
	ldrhw r1	`1111 1111 0001`	$res(31:16) \rightarrow r1$
	sqrt r1	`1111 1111 0010`	$\sqrt{r1} \rightarrow r1$
	prim r1	`1111 1111 0011`	$r1 \in \mathbb{P}\ ?\ 1 \rightarrow pr : 0 \rightarrow pr$
$\mathcal{F}$	rand	`1111 1111 1111 0000`	create a random number in res

2.3.1 unused opcodes of co-processor

There is space left for 35 more opcodes:

2-register opcodes (9) : from `11110110` to `11111110` (inclusively)

1-register opcodes (11) : from `111111110100` to `111111111110` (inclusively)

0-register opcodes (15) : from `1111111111110001` to `1111111111111111` (inclusively)

2.4 Notes

- Apart from the standard operators, the following notation is used in the table above:
  - $\ll$, $\ggg$, $\gg$are shifting operators, with semantics as in Java
  - $[rx]$ means accessing memory location that is the content of register x
  - $rx(a:b)$ means the slice from a down to b in register x
  - $x{:}y$ means concatenating x and y, thus forming a 32 bit value

3 Pipelining

We are considering a 4-stage pipeline:

1. instruction fetch
2. instruction decode
3. execute $\vee$ data fetch
4. write back

4 Versions Of This Document

2007-10-13: first version **0.1**
2007-10-14: corrected first version **0.1**