marca - McAdam's RISC Computer Architecture Instruction Set Architecture

Kenan Bilic, Roland Kammerer, Wolfgang Puffitsch

February 2, 2007

1 General

- 16 16-bit registers, r0 ... r15
- any register as return address
- flags: Z, C, V, N
  - 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 are written where meaningful: P and 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)
- separate registers for interrupt vectors - read and written through "ldvec" / "stvec"
- Some parts come from the Alpha architecture. The handling of branches is inspired by the Intel x86.
- External hardware modules shall be mapped to the highest memory locations.

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 (which is an issue when the address space is limited to 64k), and the program code can be read from a ROM directly. A transient failure thus cannot destroy the program by overwriting its code section.

2 Instruction Set

Instruction	Opcode	Semantics
add r1, r2, r3	`0000`	$r1 + r2 \rightarrow r3$
sub r1, r2, r3	`0001`	$r1 - r2 \rightarrow r3$
addc r1, r2, r3	`0010`	$r1 + r2 + C \rightarrow r3$
subc r1, r2, r3	`0011`	$r1 - r2 - C \rightarrow r3$
and r1, r2, r3	`0100`	$r1 \wedge r2 \rightarrow r3$
or r1, r2, r3	`0101`	$r1 \vee r2 \rightarrow r3$
xor r1, r2, r3	`0110`	$r1 \oplus r2 \rightarrow r3$
mul r1, r2, r3	`0111`	$r1 * r2 \rightarrow r3$
div r1, r2, r3	`1000`	$r1 \div r2 \rightarrow r3$
udiv r1, r2, r3	`1001`	$r1 \div r2 \rightarrow r3$, unsigned
ldil r1, n8	`1010`	$(r1 \wedge \texttt{0xff00}) \vee n8 \rightarrow r1, -128 \leq n8 \leq 255$
ldih r1, n8	`1011`	$(r1 \wedge \texttt{0x00ff}) \vee (n8 \ll 8) \rightarrow r1, -128 \leq n8 \leq 255$
ldib r1, n8	`1100`	$n8 \rightarrow r1, -128 \leq n8 \leq 127$
mov r1, r2	`11010000`	$r2 \rightarrow r1$
mod r1, r2	`11010001`	$r1 \bmod r2 \rightarrow r1$
umod r1, r2	`11010010`	$r1 \bmod r2 \rightarrow r1$, unsigned
not r1, r2	`11010011`	$\neg r2 \rightarrow r1$
neg r1, r2	`11010100`	$-r1 \rightarrow r2$
cmp r1, r2	`11010101`	$r1 - r2$, sets flags
addi r1, n4	`11010110`	$r1 + n4 \rightarrow r1, -8 \leq n4 \leq 7$
cmpi r1, n4	`11010111`	$r1 - n4$, sets flags, $-8 \leq n4 \leq 7$
shl r1, r2	`11011000`	$r1 \ll r2 \rightarrow r1$
shr r1, r2	`11011001`	$r1 \ggg r2 \rightarrow r1$
sar r1, r2	`11011010`	$r1 \gg r2 \rightarrow r1$
rolc r1, r2	`11011011`	$(r1 \ll r2) \vee (C \ll (r2 - 1)) \vee (r1 \ggg (16 - r2 - 1))$
rorc r1, r2	`11011100`	$(r1 \ggg r2) \vee (C \ll (16 - r2)) \vee (r1 \ll (16 - r2 - 1))$
bset r1, n4	`11011101`	$r1 \vee (1 \ll n4) \rightarrow r1, 0 \leq n4 \leq 15$
bclr r1, n4	`11011110`	$r1 \wedge \neg(1 \ll n4) \rightarrow r1, 0 \leq n4 \leq 15$
btest r1, n4	`11011111`	$(r1 \ggg n4) \wedge 1 \rightarrow Z, 0 \leq n4 \leq 15$
load r1, r2	`11100000`	$[r2]{:}[r2 + 1] \rightarrow r1$
store r1, r2	`11100001`	$r1 \rightarrow [r2]{:}[r2 + 1]$
loadl r1, r2	`11100010`	$(r1 \wedge \texttt{0xff00}) \vee [r2] \rightarrow r1$
loadh r1, r2	`11100011`	$(r1 \wedge \texttt{0x00ff}) \vee ([r2] \ll 8) \rightarrow r1$
loadb r1, r2	`11100100`	$[r2] \rightarrow r1$, signed
storel r1, r2	`11100101`	$(r1 \wedge \texttt{0x00ff}) \rightarrow [r2]$
storeh r1, r2	`11100110`	$(r1 \ggg 8) \rightarrow [r2]$
call r1, r2	`11101000`	$r1 \rightarrow pc, pc \rightarrow r2$
br n8	`11110000`	$pc + n8 \rightarrow pc, -128 \leq n8 \leq 127$
brz n8	`11110001`	$Z = 1 \Rightarrow pc + n8 \rightarrow pc, -128 \leq n8 \leq 127$
brnz n8	`11110010`	$Z = 0 \Rightarrow pc + n8 \rightarrow pc, -128 \leq n8 \leq 127$
brle n8	`11110011`	$(Z = 1) \vee (N \neq V) \Rightarrow pc + n8 \rightarrow pc, -128 \leq n8 \leq 127$
brlt n8	`11110100`	$(Z = 0) \wedge (N \neq V) \Rightarrow pc + n8 \rightarrow pc, -128 \leq n8 \leq 127$
brge n8	`11110101`	$(Z = 1) \vee (N = V) \Rightarrow pc + n8 \rightarrow pc, -128 \leq n8 \leq 127$
brgt n8	`11110110`	$(Z = 0) \wedge (N = V) \Rightarrow pc + n8 \rightarrow pc, -128 \leq n8 \leq 127$
brule n8	`11110111`	$(Z = 1) \vee (C = 1) \Rightarrow pc + n8 \rightarrow pc, -128 \leq n8 \leq 127$
brult n8	`11111000`	$(Z = 0) \wedge (C = 1) \Rightarrow pc + n8 \rightarrow pc, -128 \leq n8 \leq 127$
bruge n8	`11111001`	$(Z = 1) \vee (C = 0) \Rightarrow pc + n8 \rightarrow pc, -128 \leq n8 \leq 127$
brugt n8	`11111010`	$(Z = 0) \wedge (C = 0) \Rightarrow pc + n8 \rightarrow pc, -128 \leq n8 \leq 127$
sext r1, r2	`11111011`	$(r1 \ll 8) \gg 8 \rightarrow r2$
ldvec r1, n4	`11111100`	interrupt vector $n4 \rightarrow r1, 0 \leq n4 \leq 15$

stvec r1, n4	`11111101`	$r1 \rightarrow$ interrupt vector $n4, 0 \leq n4 \leq 15$
jmp r1	`111111100000`	$r1 \rightarrow pc$
jmpz r1	`111111100001`	$Z = 1 \Rightarrow r1 \rightarrow pc$
jmpnz r1	`111111100010`	$Z = 0 \Rightarrow r1 \rightarrow pc$
jmple r1	`111111100011`	$(Z = 1) \vee (N \neq V) \Rightarrow r1 \rightarrow pc$
jmplt r1	`111111100100`	$(Z = 0) \wedge (N \neq V) \Rightarrow r1 \rightarrow pc$
jmpge r1	`111111100101`	$(Z = 1) \vee (N = V) \Rightarrow r1 \rightarrow pc$
jmpgt r1	`111111100110`	$(Z = 0) \wedge (N = V) \Rightarrow r1 \rightarrow pc$
jmpule r1	`111111100111`	$(Z = 1) \vee (C = 1) \Rightarrow r1 \rightarrow pc$
jmpult r1	`111111101000`	$(Z = 0) \wedge (C = 1) \Rightarrow r1 \rightarrow pc$
jmpuge r1	`111111101001`	$(Z = 1) \vee (C = 0) \Rightarrow r1 \rightarrow pc$
jmpugt r1	`111111101010`	$(Z = 0) \wedge (C = 0) \Rightarrow r1 \rightarrow pc$
intr n4	`111111101011`	interrupt vector $n4 \rightarrow pc, pc \rightarrow ira, flags \rightarrow shflags, 0 \leq n4 \leq 15$
getira r1	`111111101100`	$ira \rightarrow r1$
setira r1	`111111101101`	$r1 \rightarrow ira$
getfl r1	`111111101110`	$flags \rightarrow r1$
setfl r1	`111111101111`	$r1 \rightarrow flags$
getshfl r1	`111111110000`	$shflags \rightarrow r1$
setshfl r1	`111111110001`	$r1 \rightarrow shflags$
reti	`1111111111110000`	$ira \rightarrow pc, shflags \rightarrow flags$
nop	`1111111111110001`	do nothing
sei	`1111111111110010`	$1 \rightarrow I$
cli	`1111111111110011`	$0 \rightarrow I$
error	`1111111111111111`	invalid operation

2.1 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
  - $[x]$ means accessing memory location x, 8 bits wide
  - $x{:}y$ means concatenating x and y, in the sense of forming a 16-bit value from two 8-bit values
- Modulo does not follow the patterns for "div" and "udiv", because there was not enough room for two more 3-operand operations. The assembler accepts the mnemonic with 3 registers as operands and substitute it with the according "mov" and "mod" instructions.

2.2 Instruction formats

The following formats for instructions are to be used:

Bits 15 ... 12	Bits 11 ... 8	Bits 7 ... 4	Bits 3 ... 0
Opcode	r3	r2	r1
Opcode	n8		r1
Opcode		r2	r1
Opcode		n4	r1
Opcode		n8	
Opcode			r1
Opcode			n4
Opcode			

3 Versions Of This Document

2006-10-04: Draft version **0.1**
2006-10-05: Draft version **0.2**

- rearrangement of some ops

2006-10-11: Draft version **0.3**

- replaced "ror"/"rol" with "mod"/"umod"
- refined considerations of direction flag
- proposal for priorities of implementation

2006-10-28: Draft version **0.4**

- settled to singed loads
- settled to shifts by registers
- dropped "push"/"pop"; the secondary result would cause a considerable overhead
- specified pipelining

2006-10-30: Draft version **0.5**

- added shflags to ease interrupt (and stack) handling
- a few refinements

2006-12-02: Draft version **0.6**

- the first register is the target with "mov" and "not" now.
- now the second register is always the address when accessing memory
- reversed order with immediate loads
- "ldvec" and "stvec" use the same order now
- fixed instruction format for immediate loads

2006-12-14: Draft version **0.7**

- dropped "ldpgm" in favor of a ROM which is mapped to the ordinary memory space
- moved section about pipelinign to the implementation document
- removed note about interrupts; they are implemented already