U5 Architecture Manual

Revision 0.3

March 25th 2017

Contents

1 Introduction

The U5 architecture is a 32 bit big endian fixed instruction length RISC architecture.

It has 64 full size registers, of which 62 are general purpose, and 1MiB ($2^{20}B$) of volatile memory. It can access one storage unit with a size of up to 2TiB ($2^{41}B$).

A console can be attached for user interaction.

2 Registers

The architecture has 64 32 bit registers named `r0-63`. Of those, two have special semantics:

- Register 0 always holds the value `0`. Any operation modifying register 0 will silently fail (i.e. it has no effect).
- Register 63 is the program counter. See Section 5.

No other register carries any special meaning from the CPU's perspective, although these conventions are usually followed:

- Registers 1 and 2 are used for function return values.
- Registers 3 through 9 are used for function arguments.
- Registers 10 through 29 are callee saved registers.
- Registers 30 through 54 are caller saved (i.e. scratch) registers.
- Registers 55 and 56 are reserved for operating systems.
- Registers 57 and 58 are reserved for assemblers.
- Register 59 is the link register.
- Register 60 is the frame pointer.
- Register 61 is the global pointer.
- Register 62 is the stack pointer.

Registers are usually referred to by an alias. See Appendix A for a common list.

The value of a register is written as $\langle * \rangle$, e.g. the value of the program counter can be written as $\langle$`r63`$\rangle$.

3 Endianness

U5 is a big endian architecture. A multi-byte entity, such as a machine word, when stored in memory, will have its most significant byte at the lowest address.

For example, the 32 bit value `0x44332211` will be stored in memory as the four bytes `0x44`, `0x33`, `0x22`, `0x11`, in that order.

4 Storage

Attached storage is accessed by sectors. One sector consists of 512 bytes. With 2^{32} addressable sectors the maximal theoretical storage capacity is 2TiB.

5 Program execution

Program execution proceeds as follows:

1. Fetch and decode the instruction located at address $\langle$`r63`$\rangle$.
2. Increment `r63` by the 4 (the length of an instruction).
3. Execute the decoded instruction.
4. Go to 1.

6 Instructions

6.1 Formats

All instructions consists of 32 bits. In the following, binary numbers are written with the most significant bit to the left. E.g. the 11 bit number 1337 (decimal) is written as `10100111001`.

There are 17 operations using two instruction formats; R-format and I-format. There is only one I-format operation, namely `MOVI`.

6.1.1 R-format

R-instructions take the form:

```
ooooooaaaaaabbbbbbcccccc----------
```

Where `ooooo` encodes the instruction operation (opcode), `aaaaaa`, `bbbbbb` and `cccccc` encodes register numbers and `----------` are unused. The register numbers encoded by `aaaaaa`, `bbbbbb` and `cccccc` are referred to as A, B and C, respectively.

6.1.2 I-format

I-instructions take the form:

```
10000rrrrrriiiiiiiiiiiiiiiisssss
```

Where `10000` are literal bits (one can think of an I-instruction as having opcode 16), `rrrrrr` encode a register number and `iiiiiiiiiiiiiiii` and `sssss` encode arbitrary values. The numbers encoded by `rrrrrr`, `iiiiiiiiiiiiiiii` and `sssss` are referred to as R, I and S, respectively.

6.2 Semantics

All arithmetic and logic functions are performed on and with 32 bit unsigned integers.

For example: a `LOADB`-instruction computes a source address as the sum of (the values of) two registers. If that sum is larger than $2^{32} - 1$ then the most significant bit is silently dropped.

A byte is 8 bits, a half word is 16 bits and a word is 32 bits.

6.2.1 `LOADB`

Format: R, Opcode: 0

Copy the byte at address $\langle B \rangle + \langle C \rangle$ to register A, zero extending to 32 bits.

6.2.2 `LOADH`

Format: R, Opcode: 1

Copy the half word at address $\langle B \rangle + \langle C \rangle$ to register A, zero extending to 32 bits.

6.2.3 `LOADW`

Format: R, Opcode: 2

Copy the word at address $\langle B \rangle + \langle C \rangle$ to register A.

6.2.4 `STOREB`

Format: R, Opcode: 4

Store the lower byte of $\langle A \rangle$ at address $\langle B \rangle + \langle C \rangle$.

6.2.5 `STOREH`

Format: R, Opcode: 5

Store the lower half word of $\langle A \rangle$ at address $\langle B \rangle + \langle C \rangle$. The destination address must be aligned to two bytes.

6.2.6 `STOREW`

Format: R, Opcode: 6

Store $\langle A \rangle$ at address $\langle B \rangle + \langle C \rangle$. The destination address must be aligned to four bytes.

6.2.7 `ADD`

Format: R, Opcode: 8

Set register A to $\langle B \rangle + \langle C \rangle$.

6.2.8 `MUL`

Format: R, Opcode: 9

Set register A to $\langle B \rangle \cdot \langle C \rangle$.

6.2.9 `DIV`

Format: R, Opcode: 10

Set register A to $\left\lfloor \frac{\langle B \rangle}{\langle C \rangle} \right\rfloor$. The instruction's behavior is undefined if $\langle C \rangle$ is 0.

6.2.10 `NOR`

Format: R, Opcode: 11

Set register A to $\sim \big(\langle B \rangle \mid \langle C \rangle \big)$. Here $\sim$ means bitwise complement and $\mid$ means bitwise or.

6.2.11 `MOVI`

Format: I

Set register R to $I \cdot 2^S$.

6.2.12 `CMOV`

Format: R, Opcode: 18

If $\langle C \rangle$ is 0 then this instruction has no effect. Otherwise set register A to $\langle B \rangle$.

6.2.13 `IN`

Format: R, Opcode: 24

Read one character from the console and set register A to its value according to the ASCII table, zero extending to 32 bits.

6.2.14 `OUT`

Format: R, Opcode: 25

Write the character in the ASCII table entry given by the lower byte of $\langle A \rangle$ to the console.

6.2.15 `READ`

Format: R, Opcode: 26

Copy storage sector $\langle B \rangle$ to volatile memory at address $\langle A \rangle$. The destination address must be aligned to 512 bytes.

6.2.16 `WRITE`

Format: R, Opcode: 27

Copy 512 bytes starting at memory address $\langle A \rangle$ to storage sector $\langle B \rangle$. The source address must be aligned to 512 bytes.

6.2.17 `HALT`

Format: R, Opcode: 31

Halt the processor. No further instructions are executed.

7 Boot sequence

At startup the following happens:

- All registers are set to 0.

- The first sector (i.e. 512 bytes) of the attached storage is copied to address 0.
- Instruction execution begins.

In particular the contents of volatile memory may be initialized arbitrary.

8 Software emulation

A naive emulation of the U5 architecture is easily modelled in most common programming languages. Companion files `skeleton.py` and `skeleton.c` distributed with this document give partial implementations in Python and C.

Using more sophisticated approaches (e.g. JIT compilation) a software emulator can achieve quite tolerable execution speeds.

A Register aliases

These register aliases are commonly used by assemblers:

Register	Alias	Register	Alias	Register	Alias	Register	Alias
r0	zz	r16	s6	r32	t2	r48	t18
r1	v0	r17	s7	r33	t3	r49	t19
r2	v1	r18	s8	r34	t4	r50	t20
r3	a0	r19	s9	r35	t5	r51	t21
r4	a1	r20	s10	r36	t6	r52	x
r5	a2	r21	s11	r37	t7	r53	y
r6	a3	r22	s12	r38	t8	r54	z
r7	a4	r23	s13	r39	t9	r55	k0
r8	a5	r24	s14	r40	t10	r56	k1
r9	a6	r25	s15	r41	t11	r57	at0
r10	s0	r26	s16	r42	t12	r58	at1
r11	s1	r27	s17	r43	t13	r59	ra
r12	s2	r28	s18	r44	t14	r60	fp
r13	s3	r29	s19	r45	t15	r61	gp
r14	s4	r30	t0	r46	t16	r62	sp
r15	s5	r31	t1	r47	t17	r63	pc