

# AXIS ETRAX 100 Programmer's Manual



# TABLE OF CONTENTS

Preface .....	7
The Background of the Axis CRIS CPU .....	9
<b>1 Architectural Description .....</b>	<b>13</b>
<b>1.1 Registers .....</b>	<b>13</b>
<b>1.2 Flags and Condition Codes .....</b>	<b>15</b>
<b>1.3 Data Organization in Memory .....</b>	<b>17</b>
<b>1.4 Instruction Format .....</b>	<b>18</b>
<b>1.5 Addressing Modes .....</b>	<b>19</b>
1.5.1 General .....	19
1.5.2 Quick immediate addressing mode .....	20
1.5.3 Register addressing mode .....	20
1.5.4 Indirect addressing mode .....	21
1.5.5 Autoincrement addressing mode .....	21
1.5.6 Immediate addressing mode .....	21
1.5.7 Indexed addressing mode .....	22
1.5.8 Indexed with assign addressing mode .....	22
1.5.9 Offset addressing mode .....	23
1.5.10 Offset with assign addressing mode .....	25
1.5.11 Double indirect addressing mode .....	26
1.5.12 Absolute addressing mode .....	27
1.5.13 Multiple addressing prefix words .....	27
<b>1.6 Branches, Jumps and Subroutines .....</b>	<b>27</b>
1.6.1 Conditional branch .....	27
1.6.2 Jump instruction .....	29
1.6.3 Implicit jumps .....	29
1.6.4 Switches and table jumps .....	30
1.6.5 Subroutines .....	32
1.6.6 New Subroutine instructions in the ETRAX 100 .....	33
<b>1.7 Interrupts .....</b>	<b>34</b>
1.7.1 NMI .....	36
<b>1.8 Software Breakpoints .....</b>	<b>36</b>
<b>1.9 Hardware Breakpoint Mechanism .....</b>	<b>36</b>
<b>1.10 Multiply and Divide .....</b>	<b>37</b>
1.10.1 General .....	37
1.10.2 Multiply .....	37
1.10.3 Divide .....	37
<b>1.11 Extended Arithmetic .....</b>	<b>38</b>
<b>1.12 Reset .....</b>	<b>39</b>
1.12.1 Normal case .....	39
<b>1.13 Version Identification .....</b>	<b>40</b>
<b>2 Instruction Set Description .....</b>	<b>41</b>
<b>2.1 Definitions .....</b>	<b>41</b>
<b>2.2 Instruction Set Summary .....</b>	<b>42</b>

## Table of Contents

---

2.2.1	Size Modifiers .....	42
2.2.2	Addressing Modes .....	43
2.2.3	Data Transfers .....	44
2.2.4	Arithmetic Instructions .....	44
2.2.5	Logical Instructions .....	45
2.2.6	Shift Instructions .....	46
2.2.7	Bit Test Instructions .....	46
2.2.8	Condition Code Manipulation Instructions .....	46
2.2.9	Jump and Branch Instructions .....	47
2.2.10	No Operation Instruction .....	47
<b>2.3</b>	<b>Instruction Format Summary .....</b>	<b>48</b>
2.3.1	Summary of Quick Immediate Mode Instructions .....	48
2.3.2	Summary of Register Instructions with Variable Size .....	49
2.3.3	Summary of Register Instructions with Fixed Size .....	50
2.3.4	Summary of Indirect Instructions with Variable Size .....	51
2.3.5	Summary of Indirect Instructions with Fixed Size .....	52
<b>2.4</b>	<b>Addressing mode prefix formats .....</b>	<b>53</b>
<b>2.5</b>	<b>Instructions in Alphabetical Order .....</b>	<b>55</b>
	ABS .....	56
	ADD 2-operand .....	57
	ADD 3-operand .....	58
	ADDI .....	59
	ADDQ .....	60
	ADDS 2-operand .....	61
	ADDS 3-operand .....	62
	ADDU 2-operand .....	63
	ADDU 3-operand .....	64
	AND 2-operand .....	65
	AND 3-operand .....	66
	ANDQ .....	67
	ASR .....	68
	ASRQ .....	69
	AX .....	70
	Bcc .....	71
	BOUND 2-operand .....	73
	BOUND 3-operand .....	75
	BREAK .....	76
	BTST .....	77
	BTSTQ .....	78
	CLEAR .....	79
	CLEARF .....	80
	CMP .....	81
	CMPQ .....	82
	CMPS .....	83
	CMPU .....	84
	DI .....	85
	DSTEP .....	86
	EI .....	87
	JBRC .....	88
	JIR .....	90
	JIRC .....	91
	JSR .....	93
	JSRC .....	95
	JUMP .....	97
	LSL .....	98

---

LSLQ .....	99
LSR .....	100
LSRQ .....	101
LZ .....	102
MOVE from s to Rd.....	103
MOVE from Rs to memory.....	104
MOVE to Pd.....	105
MOVE from Ps .....	106
MOVEM from memory.....	107
MOVEM to memory.....	108
MOVEQ .....	109
MOVS .....	110
MOVU .....	111
MSTEP .....	112
NEG .....	113
NOP .....	114
NOT .....	115
OR 2-operand .....	116
OR 3-operand .....	117
ORQ .....	118
POP to Rd.....	119
POP to Pd .....	120
PUSH from Rs.....	121
PUSH from Ps.....	122
RET .....	123
RETB .....	124
RETI .....	125
Scc .....	126
SETF .....	128
SUB 2-operand .....	129
SUB 3-operand .....	130
SUBQ .....	131
SUBS 2-operand .....	132
SUBS 3-operand .....	133
SUBU 2-operand .....	134
SUBU 3-operand .....	135
SWAP .....	136
TEST .....	138
XOR .....	140
<b>3 CRIS Execution Times .....</b>	<b>141</b>
<b>3.1 Introduction .....</b>	<b>141</b>
<b>3.2 Instruction Execution Times .....</b>	<b>141</b>
<b>3.3 Complex Addressing Modes Execution Times .....</b>	<b>143</b>
<b>3.4 Interrupt Acknowledge Execution Time .....</b>	<b>143</b>
<b>4 Assembly Language Syntax .....</b>	<b>145</b>
<b>4.1 General .....</b>	<b>145</b>
<b>4.2 Definitions .....</b>	<b>145</b>
<b>4.3 Files, Lines and Fields .....</b>	<b>146</b>
<b>4.4 Labels and Symbols .....</b>	<b>146</b>
<b>4.5 Opcodes .....</b>	<b>147</b>
<b>4.6 Operands .....</b>	<b>147</b>

## Table of Contents

---

4.6.1	General .....	147
4.6.2	Expressions .....	147
<b>4.7</b>	<b>Addressing Modes .....</b>	<b>149</b>
<b>4.8</b>	<b>Assembler Directives .....</b>	<b>152</b>
4.8.1	Directives controlling the storage of values .....	152
4.8.2	Directives controlling storage allocation .....	153
4.8.3	Symbol handling .....	154
<b>4.9</b>	<b>Alignment .....</b>	<b>155</b>
<b>5</b>	<b>CRIS COMPILER SPECIFICS .....</b>	<b>157</b>
<b>5.1</b>	<b>CRIS Compiler Options .....</b>	<b>157</b>
<b>5.2</b>	<b>CRIS Preprocessor Macros .....</b>	<b>158</b>
<b>5.3</b>	<b>The CRIS ABI .....</b>	<b>159</b>
5.3.1	Introduction .....	159
5.3.2	CRIS GCC fundamental data types .....	159
5.3.3	CRIS GCC object memory layout .....	160
5.3.4	CRIS GCC calling convention .....	161
5.3.5	Stack frame layout .....	162
<b>6</b>	<b>The ETRAX 4 .....</b>	<b>163</b>
<b>6.1</b>	<b>Introduction .....</b>	<b>163</b>
<b>6.2</b>	<b>Special Registers .....</b>	<b>163</b>
<b>6.3</b>	<b>Flags and Condition Codes .....</b>	<b>164</b>
<b>6.4</b>	<b>Data Organization in Memory .....</b>	<b>165</b>
<b>6.5</b>	<b>Branches, Jumps and Subroutines .....</b>	<b>166</b>
<b>6.6</b>	<b>Interrupts and Breakpoints in the ETRAX 4 .....</b>	<b>166</b>
<b>6.7</b>	<b>Reset in the ETRAX 4 .....</b>	<b>167</b>
6.7.1	Normal case .....	167
6.7.2	Automatic program download .....	167
<b>6.8</b>	<b>DMA .....</b>	<b>168</b>
6.8.1	The ETRAX 4 DMA .....	168
<b>6.9</b>	<b>Instruction Set .....</b>	<b>168</b>
6.9.1	Differences in the instructions .....	168
<b>6.10</b>	<b>Execution Times for the ETRAX 4 .....</b>	<b>169</b>
6.10.1	Introduction .....	169
6.10.2	Instruction execution times .....	169
6.10.3	Complex addressing modes execution times .....	171
6.10.4	Interrupt acknowledge execution time .....	172
6.10.5	DMA transfer execution time .....	172

## Preface

Our goal in developing the ETRAX 100 is to have a single chip solution for peripheral server applications on a Fast Ethernet. It is used in the AXIS ThinServerTechnology, but also enables designers to build embedded servers with an excellent price/performance ratio required by the growing market of network and web appliances.

### About Axis

Axis Communications is dedicated to providing innovative solutions for network-connected computer peripherals. Since the company started in 1984, Axis has been one of the fastest growing companies in the market, and is now a leader in its field.

**ThinServer™ Technology** - The core of all Axis' products, ThinServer™ technology enables our products to act as intelligent file server independent ThinServer™ devices. A ThinServer™ device is a network server which includes "thin" embedded server software capable of simultaneous multiprotocol communication, scalable RISC hardware, and a built-in Web server which allows easy access and management via any standard Web browser. ThinServer™ technology makes it possible to connect any electronic device to the network, thus providing "Access to everything".

Today, Axis Communications is offering ThinServer™ technology as well as six major ThinServer™ product lines consisting of:

**Network Print Servers** - offer you a powerful and cost-efficient method for sharing printer resources in your network. They connect to any standard printer, featuring high performance, simple management, and easy upgrading across the network. The print servers are available in Ethernet, Fast Ethernet and Token Ring versions.

**IBM Mainframe and S/3x - AS/400 Print Servers and Protocol Converters** -

includes a wide range of LAN, coax and twinax attached print servers for the IBM host environment. By emulating IBM devices, these servers provide conversion of the IPDS, SCS, and 3270DS data streams to the major ASCII printer languages.

**Network Attached Optical Media Servers** - provide you with a flexible and cost-efficient solution for sharing CD-ROMs, DVD-ROMs, and other optical media across the network. They are available in Ethernet, Fast Ethernet and Token Ring versions.

**Network Attached Storage Servers** - offer network connectivity for re-writable media such as hard disks and Iomega Jaz cartridges, which via the storage server, can be backed up on DAT tapes. They are only available in Ethernet versions.

**Network Camera Servers** - provide live images using standard Internet technology, thus enabling access to live cameras via any standard Web browser. They offer a perfect solution for remote surveillance over the Internet, and their sharp images can bring life into any web site. These servers support Ethernet as well as PSTN and GSM phone lines.

**Network Scan Servers** - enable easy distribution of paper-based information across workgroups and the enterprise. By sending the scanned documents to your

destination via the Internet/intranet, you will reduce your faxing/mailing costs, as well as save time, thus improving your organization efficiency.

### Support services

Should you require any technical assistance, please contact your Axis dealer. If they can not answer you questions immediately, your Axis dealer will forward your queries through the appropriate channels to ensure you a rapid response.

If you are connected to the Internet, you will find on-line manuals, technical support, firmware updates, application software, company information, on the addresses listed below.

	<a href="http://www.axis.com">http://www.axis.com</a>
<b>WWW:</b>	<a href="http://www.se.axis.com">http://www.se.axis.com</a> <a href="http://www.developer.axis.com">http://www.developer.axis.com</a>
<b>FTP server:</b>	<a href="ftp://ftp.axisinc.com/pub/axis">ftp://ftp.axisinc.com/pub/axis</a> <a href="ftp://ftp.axis.com/pub/axis">ftp://ftp.axis.com/pub/axis</a>
<b>Support email address:</b>	<a href="mailto:tech-support@axis.com">tech-support@axis.com</a>

## The Background of the Axis CRIS CPU

Axis started joint development of the CRIS CPU in 1990, together with Lund University located in Lund, Sweden. The university specializes in RISC processor design and is renowned for advanced studies in VLSI and HDL technology. The reason Axis initiated the CRIS design was due to the lack of a suitable standard product. The requirements of an acceptable standard product were:

- Low power consumption
- High processing power
- Very small minimal system
- High level of integration
- Scalability, allowing designs to be either high-performance or cost efficient
- 32 bit CPU with 16 bit instruction format
- Low radiation, EMI
- Compact code, although 32-bit RISC power

After looking at the standard processors available at that time such as the AMD 29200, ARM6, and Intel 80960CA, we could not find a processor that met all these requirements and decided, therefore, to develop our own processor. Our approach is similar to that for the PDA processors (like ARM, etc). However, for the Axis system an integrated core was needed instead of just a standard component.

Normally the development of a processor is a lengthy project that involves several millions of dollars, typically an architecture costs in the range of \$100 M USD. By using HDL languages, modern design tools, and the knowledge from Lund University, we significantly reduced both the development lead time and cost. Using commercially available chip design tools, this project would not have been feasible.

### The CRIS name

CRIS is an acronym for *Code Reduced Instruction Set* and is an anagram of RISC. The name was invented by one of the design engineers.

### The C compiler

After deciding to design a processor, the next challenge was to design the compiler. At this time all our software was written in the C language, so we needed a C compiler. We decided, therefore, to make our own version of the Free Software Foundation GNU C compiler, which is a very well known compiler used by various microprocessor vendors.

The CRIS compiler project became a big success producing both compact and optimized code. As one of the Axis' compiler design engineers put it, "The only way to make a truly optimized compiler is to optimize the CPU to the compiler." This was the key element and a unique design approach to include the design of the CPU into the design of the compiler.

### The CRIS instruction set

When designing a CPU, one usually designs the architecture, selects bit fields, and decides how to use addressing modes etc., and adapts the compiler to the CPU. However, for the CRIS CPU we decided to do it the other way around. Instead of starting with the processor, we started by looking at the compiler, or more specifically, the code generator of the compiler.

We wrote a C-based CPU simulator, which was easy to adapt to the new specifications of a CPU, and compiled large pieces of C code using the GNU C compiler. We evaluated numerous permutations of the code generator to find an optimal instruction set match for the GNU compiler.

After six months and several iterations, we thought we had found the ideal instruction set. The result was that our CRIS processor requires 10% less executable code than a traditional CISC processor (the NS 32CG16, using the National GNX compiler). Normally a RISC processor requires more executable code space than a CISC.

The main concepts derived from the iterations with the compiler are:

- Using addressing mode prefix bytes rather than a separate addressing mode field eliminates the need for 3-address instructions. In fact, we discovered that the form we use is the only one required, see later on for details.
- The benefit of having just 14 generic + 2 special purpose (stack pointer and program counter) registers. We ran extensive simulations where we increased/decreased the amount of registers. The results surprised us. We thought there was a close connection between performance and the number of registers.
- Including advanced addressing modes normally not available in a true RISC CPU can be done without sacrificing bit fields. This was implemented in the compiler.
- The design where the instruction format is 16 bits, but the processor is a true 32 bit processor. All register sizes may be used as 8, 16 or 32 bits. This saves a lot of memory compared to a 32 bit instruction CPU.
- There was no need for a link instruction or higher software level instructions.
- Move multiple instruction. With this instruction we can push/pop various registers (in order). We also discovered that the limitation that the registers had to be in order did not matter.

We also analyzed speed optimization. Most of the code size optimization ideas also improved speed. Some more speed specific features are:

- Call to or return from subroutines. The traditional approach is to push the return address at the call, but we use a subroutine return pointer instead. This means the lowest level (and most often called) subroutines do not need to push/pop the stack. In some applications, this may result in a performance gain of 20%.
- Quick instruction format. Most common arithmetics with an immediate value of 5 bits (which is very common in most programs) will only take one instruction cycle to execute. This saves speed and code size. We found that the 5 bits (+ 1 sign bit) was an ideal compromise after extensive simulations.

- A common need is clear memory and to test memory for a zero condition. We added these instructions.
- Efficient use of delay slots. Saves speed and code.

We had to make several compromises. Initially we wanted to have full multiplication/division and a bit blt. By running several typical applications we saw that the need for multiplication/division is not a must.

We lowered the level of support for multiplication and made it a multiply by step instruction. By using this procedure, we can do a multiplication subroutine that takes 62 cycles to execute. Also, the GNU C compiler discovers all easy multiplications (i.e. multiply by 5 is the same thing as shifting 2 steps with the barrel shifter and an add, which is only one instruction for CRIS). A similar approach was used for division.

The bit blt simply had to be dropped. It would require too much silicon, which was already scarce.

## Summary

With the ETRAX chip, incorporating the CRIS CPU now available in silicon, we have been able to evaluate how well we managed to reach our targets. The result is that we achieved all the targets except for multiplication and division. Also, the power consumption was less than expected, which is primarily due to the two-phase clock technology. Finally, we were really impressed by the fact that the ETRAX performs faster than planned and requires significantly less executable code. Our target was in the range of 30% more than the comparable 32CG16, but it actually turned out to be 10% less! All these features put together, we have seen that the ETRAX chip is ideal in executing multiprotocol network stacks on one chip.

