The Harris RTX 2000 Microcontroller

Tom Hand
Senior Scientist
Harris Semiconductor
Melbourne, Florida 32902

Abstract

Harris Semiconductor has developed the RTX 2000, a highly integrated 10MHz 16-bit microcontroller, for embedded applications that have demanding real-time requirements.

The RTX 2000 is a high-performance stack machine that has many of the advantages of RISC processors, but without their disadvantages. The RTX 2000 has a predictable run-time behavior because of its advanced, yet simple, architectural features.

Introduction

The RTX 2000 is a highly integrated 16-bit microcontroller from Harris Semiconductor that has been specifically designed for solving problems associated with embedded real-time systems. It has a predictable run-time behavior, a key requirement of real-time systems.

The RTX 2000 is derived from earlier generation Novix stack machines. Novix Inc. of Cupertino, CA first developed the Novix NC 4016 stack machine that directly executed 40 Forth primitives as well as 123 combinations of Forth words as single instructions. The NC 4016 chip used only 4000 gates that were built from just 16,000 CMOS transistors. Harris bought complete rights to the Novix technology and in addition added on-chip stacks and other on-chip support such as counter/timers, interrupt controllers, and a single-cycle multiplier.

The architecture of the RTX 2000 encourages programming in a structured manner. To accomplish this, subroutine calls have been implemented in such a way that they execute in one machine cycle. More remarkably, subroutine returns are normally free; that is, they take zero clock cycles to execute. As a result, application code is both extremely compact and fast.

The RTX 2000 is a stack machine; its machine language corresponds to certain sequences of Forth instructions. Because four internal buses may be active at the same time, it is often possible to execute the equivalent of several Forth instructions in a single machine cycle. This guarantees that application code executes very quickly.

This article first briefly compares the RTX 2000 with RISC processors. Next, the RTX 2000 architecture is examined by discussing its basic buses and registers. Third, the instruction set of the RTX 2000 is studied in detail. Finally, additional features of the RTX 2000 are discussed; these include interrupts, byte swapping and the single-cycle multiplier.

RTX 2000 Versus RISC Machines

The RTX 2000 incorporates most of the advantages of RISC architectures, but without their disadvantages.

RISC (Reduced Instruction Set Computers) machines have most of the characteristics listed below:
- a large set of registers
- a simple set of instructions
- execution of most instructions in a single cycle
- equal lengths of all instructions
- very few addressing modes
- no instructions that manipulate or modify the contents of memory
- no microcode
- utilization of pipelines and caches

Not every RISC machine has all of these characteristics, but most do have a significant number of them.

The RTX 2000 has most of the attributes listed above; the two items that really distinguish the RTX 2000 from the RISC machines are the first and last items. Because RISC machines have register-based architectures, a degree of complexity is added to their architectures to get higher performance. Instructions are divided into components and pipelines are introduced to make these components execute as stages in parallel. Typical pipeline stages are instruction fetch, instruction decode, instruction execute and instruction write-out. Partly because of pipeline stalls and delayed branches, caches are introduced. The resulting RISC architectures have better performance than they had before the complexity was added, but at a definite cost.

In contrast, the RTX 2000 has a stack-based architecture so that operands are implicitly known. This subtle idea is what makes stack machines simple. RISC machines, on the other hand, require explicitly referenced operands and this leads to extra overhead, both in compiling by optimizing compilers and in the size of memory required for the final compiled code.

The Architecture of the RTX 2000

The RTX 2000 is a highly integrated 16-bit microcontroller with three on-chip counter/timers, an on-chip interrupt controller, two on-chip stack controllers and single cycle 16-by-16 on-chip hardware multiplier. Figure 1 is a block diagram that illustrates the basic architecture of the RTX 2000.

![RTX 2000 Architecture Diagram](image)

Figure 1: RTX 2000 Architecture
At the center of the figure is the RTX core processor, which is the stack machine. Other portions of the RTX 2000 are connected to the core processor through one of the RTX 2000's four buses listed below:

- parameter stack bus
- return stack bus
- ASIC bus
- memory bus

As a stack machine, the RTX 2000 has two on-chip stacks that are called the parameter and return stacks. The parameter stack is used for passing arguments and manipulating data, the return stack is used for return addresses and looping arguments. Having the stacks on-chip significantly improves the performance of the RTX 2000. The parameter and return stacks are connected to the core processor through separate parameter stack and return stack buses.

The ASIC bus is a high speed bus that connects the RTX 2000 to on-chip registers and devices as well as to external I/O devices. Each device attached to the ASIC bus is assigned a specific I/O address. The RTX 2000 can address 32 ASIC devices.

The processor’s control and status registers are connected to the ASIC bus. ASIC addresses 0 through 23 are assigned to on-chip registers and devices. This includes the program counter, the square root register, the configuration register, the data page register, the code page register, the user page register, the user base register and the interrupt mask register.

In addition, the two on-chip stack controllers, the on-chip interrupt controller, the three on-chip 16-bit counter/timers and the 16-by-16 single cycle multiplier are all accessible through the ASIC bus via ASIC bus read and write instructions.

Again, one of the advanced features of the RTX 2000 is its high speed ASIC bus. This bus provides a natural extension to both internal and external devices.

Up to 512K words of memory may be addressed by the RTX 2000 through the memory bus. Memory is divided into sixteen pages, each containing 32K words of memory. Memory may consist of combinations of RAM, ROM or memory mapped I/O devices.

**RTX 2000 Internal Registers**

There are eight 16-bit registers which are internal to the processor. A brief description of these registers is given below:

**TOP** contains the top item of the parameter stack. It is called the Top Register.

**NEXT** contains the second item of the parameter stack. It is called the Next Register.

**IR** contains the instruction currently being executed. It is called the Instruction Register.

**PC** contains the address of the next instruction to be fetched from main memory. It is called the Program Counter.

**CR** contains bits that indicate the status of the RTX 2000. It is called the Configuration Register.

**I** contains the top item of the return stack. It is called the Index Register.

**MD** normally contains the divisor during multi-step math operations. It is called the Multi-step Divide Register.

**SR** normally contains intermediate values used during square root calculations. It is called the Square Root Register.
The **RTX 2000 Instruction Set**

The RTX 2000 is a 16-bit machine. All of its instructions are 16 bits wide, with the exception of long literals that take 16 bits for the instruction and 16 bits for the actual literal value.

RTX 2000 instructions execute in either one or two machine cycles. All primitive Forth words which do not perform memory accesses execute in one clock cycle. Memory access instructions require two cycles.

Most math, I/O and memory reference operations take their operands from the parameter stack and leave their results on the parameter stack.

**Instruction Format**

The general format for the RTX 2000 instructions is given below:

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>12–15</td>
<td>type of instruction</td>
</tr>
<tr>
<td>ALU</td>
<td>8–11</td>
<td>ALU function to be performed</td>
</tr>
<tr>
<td>SC</td>
<td>6–7</td>
<td>subclass, depends on the class field</td>
</tr>
<tr>
<td>;</td>
<td>5</td>
<td>return bit, causes a return when set</td>
</tr>
<tr>
<td>Data</td>
<td>0–4</td>
<td>indicates shift, short literal, ASIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bus address or memory address</td>
</tr>
</tbody>
</table>

**Instruction Class**

The four most significant bits of each instruction indicate the class type of that instruction. There are eight general types of instructions as illustrated below:

<table>
<thead>
<tr>
<th>Class</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–7</td>
<td>Subroutine call</td>
</tr>
<tr>
<td>8–9</td>
<td>Branches and loops</td>
</tr>
<tr>
<td>10</td>
<td>Math/logic functions</td>
</tr>
<tr>
<td>11</td>
<td>Register and short literal operations</td>
</tr>
<tr>
<td>12</td>
<td>User memory access</td>
</tr>
<tr>
<td>13</td>
<td>Long literals</td>
</tr>
<tr>
<td>14</td>
<td>Memory access by word</td>
</tr>
<tr>
<td>15</td>
<td>Memory access by byte</td>
</tr>
</tbody>
</table>

**Subroutine Calls**

As mentioned earlier, the RTX 2000 is optimized for writing modular applications. Subroutine calls within the same memory page are performed in one clock cycle; a call to a subroutine in a different memory page takes three clock cycles. Returns take zero clock cycles when they are performed as part of another instruction and one cycle when they are performed as separate instructions.

If an instruction has bit 15 set to 0, it is a subroutine call. The format for the subroutine call is

```
0aaa aaaa aaaa aaaa
```

where the address of the subroutine is `aaaa aaaa aaaa aaaa`, which is calculated by shifting the low-order fifteen bits to the left by one bit and inserting a 0 in the least significant bit. Note that the address of the subroutine called is embedded in the 16-bit call instruction.

**Branch and Loops**

The `0BRANCH` instruction belongs to class 8, while the `BRANCH` and `NEXT` instructions belong to class 9. These instructions perform conditional and unconditional branches, respectively.
All branches take one cycle, independent of whether or not the branch is actually taken. This is in contrast to many RISC and CISC processors which take a variable number of cycles depending on whether or not a branch is taken.

For high speed looping, the NEXT form of the conditional branch instruction may be used. With this instruction, a count is put in the Index register to indicate how many times the loop is to be performed. The NEXT instruction tests the contents of the Index register at the conclusion of each pass through the loop. If the contents of the Index register are 0, the return stack is popped and execution continues with the instruction that immediately follows NEXT; otherwise, the contents of the Index register are decremented by one and a branch back to the start of the loop is taken.

The format for this class of instructions is

\[ \text{100c cbba aaaa aaaa} \]

where the two bits ‘cc’ specify the condition for branching as indicated below:

<table>
<thead>
<tr>
<th>cc</th>
<th>Branch condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Branch if the contents of TOP is 0. Leave the stack unchanged.</td>
</tr>
<tr>
<td>01</td>
<td>Branch if the contents of TOP is 0. Pop the stack.</td>
</tr>
<tr>
<td>10</td>
<td>Perform an unconditional branch.</td>
</tr>
<tr>
<td>11</td>
<td>Branch if the contents of the INDEX register &lt;&gt; 0.</td>
</tr>
</tbody>
</table>

and the two bits ‘bb’ determine the block selection as indicated below:

<table>
<thead>
<tr>
<th>bb</th>
<th>Block selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Branch within the same memory block (no change to bits 10 – 15 of the instruction).</td>
</tr>
<tr>
<td>01</td>
<td>Branch to the next memory block (add 1 to the value in bits 10 – 15).</td>
</tr>
<tr>
<td>10</td>
<td>Branch to block 0 (set each bit in 10 – 15 to a value of 0).</td>
</tr>
<tr>
<td>11</td>
<td>Branch to the previous block (subtract one from the value in bits 10 – 15).</td>
</tr>
</tbody>
</table>

The value ‘aaaaa’ indicates the offset from the start of the new block. This value replaces bits 1 – 9 of the Program Counter. Branch addresses are always even numbers.

The RTX 2000 has a streamed instruction capability that can execute repeatedly an instruction without continually performing the fetch cycle of the instruction. This feature is very useful for fast data transfers, loops and certain math functions.

**ALU Operations**

The ALU operations are performed by the RTX’s 16-bit ALU, one operand being the contents of the TOP register and the other being determined by the instruction. The result of an ALU operation is placed in the TOP register.

The twelve possible ALU operations are briefly described below. ‘T’ indicates the TOP register and ‘Y’ indicates the second source of the ALU operation.
<table>
<thead>
<tr>
<th>cccc</th>
<th>aaa</th>
<th>Function</th>
<th>Resulting carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>001</td>
<td>T AND Y</td>
<td>no change</td>
</tr>
<tr>
<td>0011</td>
<td></td>
<td>T NOR Y</td>
<td>no change</td>
</tr>
<tr>
<td>0100</td>
<td>010</td>
<td>T – Y</td>
<td>ALU carry</td>
</tr>
<tr>
<td>0101</td>
<td></td>
<td>T – Y with borrow</td>
<td>ALU carry</td>
</tr>
<tr>
<td>0110</td>
<td>011</td>
<td>T OR Y</td>
<td>no change</td>
</tr>
<tr>
<td>0111</td>
<td></td>
<td>T NAND Y</td>
<td>no change</td>
</tr>
<tr>
<td>1000</td>
<td>100</td>
<td>T + Y</td>
<td>ALU carry</td>
</tr>
<tr>
<td>1001</td>
<td></td>
<td>T + Y with carry</td>
<td>ALU carry</td>
</tr>
<tr>
<td>1010</td>
<td>101</td>
<td>T XOR Y</td>
<td>no change</td>
</tr>
<tr>
<td>1011</td>
<td></td>
<td>T XNOR Y</td>
<td>no change</td>
</tr>
<tr>
<td>1100</td>
<td>110</td>
<td>Y – T</td>
<td>ALU carry</td>
</tr>
<tr>
<td>1101</td>
<td></td>
<td>Y – T with borrow</td>
<td>ALU carry</td>
</tr>
</tbody>
</table>

A literal is a constant value that may be used as part of either an arithmetic or a logical operation. There are two types of literals that are recognized by the RTX 2000; namely, short literals and long literals. Short literals are stored as five bits whereas long literals take sixteen bits.

**Short Literals**

Short literals are 5-bit unsigned integer values between 0 and 31. They are embedded in short literal instructions and are denoted by 'ddddd'. The format for the short literal instructions is

\[ 1011 \texttt{vvvv} \texttt{v1};d \texttt{dddd} \]

where the 'v' bits determine the particular variation of the short literal instruction. For the complete list of variations, see Table 1 which contains all the RTX 2000 instructions.

When a short literal instruction is executed, the value represented by 'ddddd' is loaded into the TOP register.

**User Memory Access**

User memory space consists of blocks of 32 words that can be accessed without having to first calculate an address and then load it into the TOP register. The location of the user memory space can be specified by the user by loading a desired base address into the user base register. The offset within the 32-word block is embedded in the user access instruction. Therefore, this class of instructions perform fast reads and writes from and to user memory space.

The offset is encoded as a 5-bit field in the instruction and is denoted by 'uuuuu'. The format for the user memory access instructions is

\[ 1100 \texttt{vvvv} \texttt{v0};u \texttt{uuuu} \]

where the 'v' bits determine the particular variation of the user memory access instruction. For the complete list of variations, see Table 1.

**Long Literals**

A long literal value may be a signed or unsigned 16-bit integer. Long literals are fetched from memory, so that an additional clock cycle is necessary for the execution of an instruction that involves a long literal.

The long literal instructions generate 16-bit values. The format for the long literal instructions is

\[ 1101 \texttt{vvvv} \texttt{v0};x \texttt{xxxx} \texttt{dddd} \texttt{dddd} \texttt{dddd} \texttt{dddd} \]
where the 'v' bits determine the particular variation of the long literal instruction and the 'x' bits are don't care bits. For the complete list of variations, see Table 1. The value dddd dddd dddd dddd is the long literal value. Long literals are the only RTX 2000 instructions that occupy two words of memory.

Data Memory Access

The RTX 2000 can access data memory as either 16-bit words or as 8-bit bytes. The format for the data memory access instructions is

```
111s vvvv vvvv
```

where the 'v' bits determine the particular variation of the data memory access instruction and the 's' bit indicates the size of the operand:

- s = 0, for 16-bit words,
- s = 1, for 8-bit bytes.

The formats for both word and byte access instructions are the same. For the complete list of variations, see Table 1.

Subroutine Returns

Instructions, that are not call or branch instructions, that have the subroutine bit (bit 5) set execute a subroutine return. The format for the subroutine return instruction is

```
1vvv vvvv vv1v vvvv
```

where the 'v' bits determine the RTX 2000 instruction to be executed along with a return. See Table 1 for the many possible variations. Note that bit 5 cannot be used as a subroutine bit in subroutine call or branch instructions since it is one of the bits used to determine the branch address.

A definite architectural advantage is gained by dedicating a single bit for indicating a subroutine return. Other architectures require 16-bit or 32-bit instructions to accomplish the same thing. For example, on the RTX 2000, the optimized code corresponding to

```
SWAP DROP DUP ;
```

is A0A0 (hex), with the return bit set to 1; whereas, the optimized code for

```
SWAP DROP DUP
```

is A080 (hex), with the return bit set to 0. In these cases, the equivalent of three or four Forth instructions are executed in one machine cycle.

The RTX 2000 Instructions

Table 1 contains the specification of the complete RTX 2000 instruction set. Each type of instruction has been discussed in an earlier section. For reference, all instructions are placed here in one table.

Interrupts

The RTX 2000 has an on-chip interrupt controller with fourteen interrupt request inputs for servicing on-chip as well as external interrupts.

Thirteen of the interrupt requests are maskable while the remaining interrupt request is non-maskable. The interrupt controller samples the interrupt request inputs during each instruction, prioritizes the active requests and signals the processor of pending interrupt requests.
Subroutine Call
0aaa aaaa aaaa aaaa call word address aaaa aaaa aaaa aaaa
Subroutine Call
1xxx xxxx xx1x xxxx return from subroutine

Single Step Math Functions
1010 000i 00;0 ssss (NOT) shift
1010 111i 00;0 ssss DROP DUP (NOT) shift
1010 cccc 00;0 ssss OVER SWAP alu-op shift.
1010 000i 01;0 ssss SWAP DROP (NOT) shift
1010 111i 01;0 ssss DROP (NOT) shift
1010 cccc 01;0 ssss alu-op shift
1010 000i 10;0 ssss SWAP DROP DUP (NOT) shift
1010 111i 10;0 ssss SWAP (NOT) shift
1010 cccc 10;0 ssss SWAP OVER alu-op shift
1010 000i 11;0 ssss DUP (NOT) shift
1010 111i 11;0 ssss OVER (NOT) shift
1010 cccc 11;0 ssss OVER OVER alu-op shift

Step Math Functions
1010 vvvv vvv1 vvvv

Branch Functions
1000 0bba aaaa aaaa ?DUP @BRANCH
1000 1bba aaaa aaaa @BRANCH
1001 0bba aaaa aaaa BRANCH
1001 1bba aaaa aaaa NEXT

Register and I/O Access
1011 000i 00;g gggg g 0g DROP (NOT)
1011 111i 00;g gggg g 0g (NOT)
1011 cccc 10;g gggg g 0g OVER alu-op
1010 000i 10;g gggg DUP g 0g (NOT)
1011 111i 10;g gggg g 0g (NOT)
1011 cccc 10;g gggg g 0g SWAP alu-op

Short Literal
1011 000i x1;d dddd d DROP (NOT)
1011 111i 01;d dddd d (NOT)
1011 cccc 01;d dddd d SWAP alu-op
1011 111i 11;d dddd d SWAP DROP (NOT)
1011 cccc 11;d dddd d SWAP alu-op

User Space 1st cycle
1100 000i 00;u uuuu u 0u SWAP
1100 111i 00;u uuuu u 0u SWAP
1100 cccc 00;u uuuu u 0u SWAP
1100 000i 10;u uuuu DUP u 1
1100 111i 10;u uuuu DUP u 1
1100 cccc 10;u uuuu u 0u SWAP
alu-op

Long Literal 1st cycle
1101 000i x0;x xxxx D SWAP
1101 111i 00;x xxxx D SWAP
1101 cccc 00;x xxxx D SWAP
1101 111i 10;x xxxx D SWAP
1101 cccc 10;x xxxx D SWAP
alu-op

Memory Access 1st cycle
111s 000i 00;x xxxx @ SWAP
111s 111i 00;x xxxx @ SWAP
111s cccc 00;x xxxx @ SWAP
111s 000p 01;x xxxx [SWAP DROP] DUP 0 SWAP
111s 111p 01;d dddd [SWAP DROP] 0 d
111s aaaa 01;d dddd [SWAP DROP] DUP 0 SWAP d SWAP alu-op
111s 000i 10;x xxxx OVER SWAP 1
111s 111i 10;x xxxx OVER SWAP 1
111s cccc 10;x xxxx @ SWAP
alu-op
111s 000p 11;x xxxx [OVER SWAP] SWAP OVER 1
111s 111p 11;d dddd [OVER SWAP] 1 d
111s aaaa 11;d dddd [OVER SWAP] SWAP OVER 1 d SWAP alu-op
alu-op

where s = 0, for memory access by word (0 and 1)
and s = 1, for memory access by byte (00 and 10)
{SWAP DROP} and {OVER SWAP} are performed if p = 0

Table 1. RTX 2000 Instructions
Interrupts can be enabled or disabled by means of the interrupt disable bit in the processor's configuration register. Interrupts are disabled when this bit is set to 1 and enabled when it is set to 0.

The RTX 2000 has a single level software interrupt capability.

**Byte Swapping**

Interfacing with non-RTX processors is supported through shared memory. The RTX 2000 has a byte swapping feature that allows 16-bit values to be read and written so that the most significant byte can be associated with either an even or an odd address.

**The Multiplier**

The on-chip single cycle hardware multiplier of the RTX 2000 multiplies two 16-bit numbers yielding a 32-bit result in only one clock cycle. The two 16-bit operands can be treated as either signed or unsigned integers. In addition, the resulting product can optionally be rounded to 16 bits. As mentioned earlier, the multiplier is connected to the ASIC bus.

**Conclusion**

In summary, the RTX 2000 is a high performance highly integrated stack machine that has been designed for embedded real-time applications. Very compact and fast code results from its advanced, yet simple, architectural features.

**References**


Dr. Tom Hand received his Ph.D. in Mathematics from the University of Oklahoma in 1972, and has been in the computer field for more than twenty years in industry and at the University. As Graduate Program Chairman in Computer Science at the Florida Institute of Technology he examined Forth as a vehicle for implementing compilers, expert systems, natural language front-ends and operating systems. Currently, he is a Senior Scientist at Harris Semiconductor and is involved with the development of the RTX family of microcontrollers.