Advance Data Sheet
December 1992

AT&T Microelectronics

ATT92010 *Hobbit*™ Microprocessor

Features

- High performance: fast response time for low power dissipation systems
  - — Single-cycle instruction execution for most instructions
  - — Peak rate of better than one cycle per instruction
  - — Operand bypass mechanism that reduces off-chip reads
  - — Branch prediction and branch folding to minimize control transfer penalties
  - — No delayed branching or load delay slot
  - — Hardware-implemented complex instructions, e.g., integer multiply and divide
- On-chip integrated resources: lower-cost hardware system design
  - — 3 Kbyte encoded instruction cache, organized as three-way set associative
  - — 256 byte stack cache which holds top of user stack
  - — 32-entry, direct-mapped, decoded instruction cache
  - — Memory management unit (MMU), with dual 32-entry translation look-aside buffers (TLBs) for text and data address translation
- Big-endian/little-endian data byte ordering: *IBM** PC and *Macintosh*† data compatibility
- Low power consumption: smaller, longer-life battery
  - — 250 mW at 3.3 V, 20 MHz with selection to higher speed
  - — 900 mW at 5.0 V, 30 MHz with selection to higher speed
  - — <50 µA in standby mode
- High code density: less RAM and lower system costs
  - — Rationalized instruction set
  - — Variable length instruction format
  - — Memory-to-memory architecture
- Low I/O traffic
  - — Integrated caches
  - — Operand bypass
  - — High code density
- Simple programming model: faster software development and lower development cost
  - — No user-visible registers
  - — Orthogonal instruction and addressing modes
  - — No code scheduling needed because of hardware hazard detection and bypassing
- Glueless integration within the AT&T 92K *Hobbit* family chip set
- IEEE 1149.1 JTAG compatible
- Advanced 0.9 µm, 2LM CMOS technology
- 132-pin JEDEC plastic quad flat package (PQFP)

* *IBM* is a registered trademark of International Business Machines Corporation.
† *Macintosh* is a trademark of Apple Computer, Inc.

Table of Contents

Contents	Page
Features	1
Description	3
Pin Information	4
Pins Grouped Functionally	5
Pins Grouped Numerically	12
Instruction Set	16
Data Types	16
Operand Addressing Modes	16
Integer Arithmetic	17
Fast Calling Sequence	19
Conditional Branches	19
Instruction Format	20
Instructions	21
Instruction (OPCODE/SUBCODE) Encodings	23
Addressing Mode (SMODE/DMODE) Encodings	25
Prefetching Strategy	26
Static Branch Prediction and Branch Folding	26
Instruction Tracing	27
Event Processing	27
Reset	27
Interrupt	28
Exceptions	29
Unimplemented Instruction	29
Trapped Niladics	30
Event Processing Priority	30
ATT92010 Stacks	30
The Stack Cache	30
Stack Cache Maintenance	31
Integer Accumulator	31
Stack Precautions	31
Control Registers	32
CONFIG—Configuration Register	32
FAULT—Fault Register	33
ID—JTAG ID Register	34
ISP—Interrupt Stack Pointer	34
MSP—Maximum Stack Pointer	35
PC—Program Counter	35
PSW—Program Status Word	36
SHAD—Shadow Register	38
SP—Stack Pointer	38
STB—Segment Table Base	39
TIMER1—Timer1 Register	40
TIMER2—Timer2 Register	40
VB—Vector Base	41
Addressing and Alignment Restrictions	42
Memory Management	42
Virtual Address Mapping	44
Segment Tables	45
Page Tables	47
Memory Management Operations	48
Bus Operation and Arbitration	49
Requesting the Bus	49
Surrendering the Bus	49
Bus Transaction Types	51
Exception Handling	54
Testability	54
Conformance	54
TAP Controller (TAPC)	54
IEEE Registers 1149.1/D5 Registers	57
Absolute Maximum Ratings	59
Handling Precautions	59
Electrical Characteristics	60
Timing Characteristics	61
Load Specifications	61
Timing Diagrams	62
Outline Diagram	68
132-pin PQFP	68

Description

The ATT92010 *Hobbit* Microprocessor is a high-performance, 32-bit RISC microprocessor implemented in the AT&T 0.9 µm double-level metal CMOS technology. The device's unusual combination of high performance, high code density, small die size, and low power consumption makes it especially well suited in battery-powered portable applications that are sensitive to system performance, weight, and cost.

Incorporation of two independent pipelines, on-chip caching, and many unique architectural features gives this device high performance with better than single-cycle peak execution, typical of RISC processors. A variable length instruction format and a memory-to-memory architecture result in high code densities, typical of the best available from CISC processors. On-chip integration of several caches, a memory management unit (MMU), and dual TLBs, together with high code density, enable development of less expensive system configurations. Such configurations require fewer support chips, less RAM, and less I/O traffic, resulting in less power dissipation and lower system cost.

This high degree of on-chip integrated system resources simplifies hardware development while the programming model simplifies system software development. Ultimately, product development is simplified and design cycle time is reduced.

Inherent in the *Hobbit* architecture is the freedom to scale on-chip caches transparently to software. This allows for continued enhancement of the device's performance both architecturally and with increased clock frequency as more advanced processing technologies become available. Furthermore, the scaling of on-chip caches can be accomplished without necessitating modification to system software and/or hardware. Thus, the user's investment in application software and hardware is better protected as the device is enhanced.

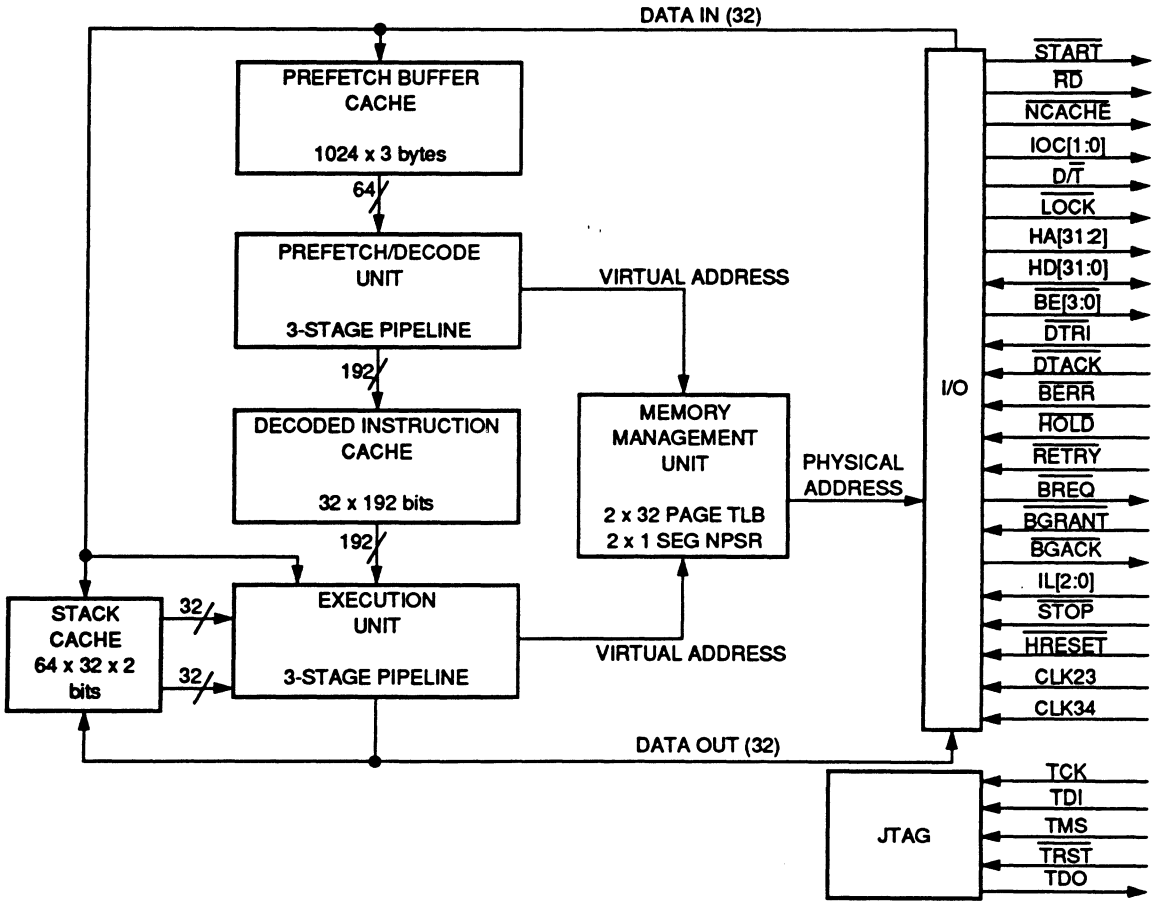


Figure 1. *Hobbit* Block Diagram

Pin Information

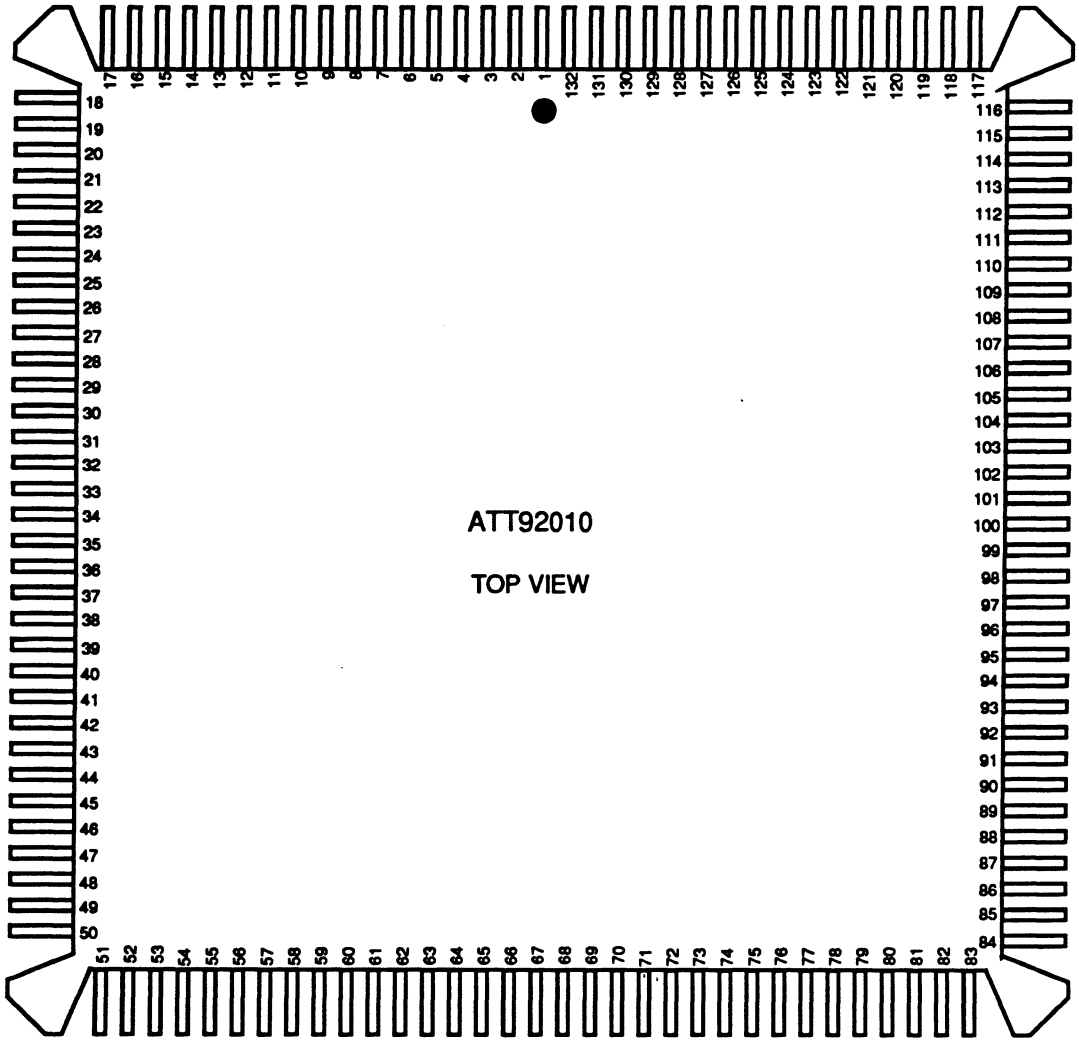


Figure 2. 132-Pin PQFP Pin Diagram

Pin Information (continued)

Pins Grouped Functionally

Table 1. Pin Descriptions

Pin	Symbol	Type*	Name/Description
57	CLK23	I	**Phase 2 3 Clock.** CLK23 is high during phases 2 and 3. The *Hobbit* microprocessor makes use of CLK23, along with CLK34, to decode a four-phase clocking system which serves as the basic reference.
55	CLK34	I	**Phase 3 4 Clock.** CLK34 is high during phases 3 and 4. The *Hobbit* microprocessor makes use of CLK34, along with CLK23, to decode a four-phase clocking system which serves as the basic reference.
56	$\overline{\text{STOP}}$	I	**Stop Clocks (Active-Low).** $\overline{\text{STOP}}$ is issued to stop the master clock decoder in phase 1. This input is asserted a setup time prior to phase 1 to halt the device.
39	$\overline{\text{BGRANT}}$	I	**Bus Grant (Active-Low).** $\overline{\text{BGRANT}}$ is used to grant exclusive use of the bus. In a multiple bus master system, only one $\overline{\text{BGRANT}}$ is to be asserted at any time to avoid bus contention. The bus arbiter asserts $\overline{\text{BGRANT}}$ to the *Hobbit* microprocessor, indicating it is bus master for the next bus transaction. While $\overline{\text{BGRANT}}$ is asserted, the *Hobbit* microprocessor remains bus master. $\overline{\text{BGRANT}}$ is asserted or deasserted by the arbiter a setup time prior to the rising edge of CLK23. When the arbiter deasserts $\overline{\text{BGRANT}}$, the *Hobbit* microprocessor relinquishes the bus after it completes the current bus transaction.
36	$\overline{\text{BREQ}}$	O	**Bus Request (Active-Low).** $\overline{\text{BREQ}}$ is asserted when the *Hobbit* microprocessor has a valid I/O transaction pending. $\overline{\text{BREQ}}$ is deasserted when there are no pending I/O transactions. The ATT92010 *Hobbit* microprocessor can deassert $\overline{\text{BREQ}}$ when there is I/O activity, but there cannot be any I/O transactions following the one in progress.
34	$\overline{\text{BGACK}}$	O	**Bus Grant Acknowledge (Active-Low).** The *Hobbit* microprocessor asserts $\overline{\text{BGACK}}$ to indicate that it has ownership of the bus. It deasserts $\overline{\text{BGACK}}$ to indicate that it has relinquished ownership of the bus.
53	$\overline{\text{HRESET}}$	I	**Reset Signal (Active-Low).** The *Hobbit* microprocessor can be reset by asserting this signal for at least two consecutive clock cycles. For multiple masters, $\overline{\text{HRESET}}$ should be synchronous to ensure proper initialization. In this mode, deassertion of $\overline{\text{HRESET}}$ should be a setup time prior to the rising edge of CLK34.

* I = input; O = output; P = power; (3) = 3-state; (R) = on-chip pull-up or pull-down resistor.

Pin Information (continued)

Pins Grouped Functionally (continued)

Table 1. Pin Descriptions (continued)

Pin	Symbol	Type*	Name/Description
42	$\overline{\text{BERR}}$	I	**Bus Error (Active-Low).** The assertion of $\overline{\text{BERR}}$ indicates an error in a bus transaction of any type. An internal I/O fault is generated when $\overline{\text{BERR}}$ is asserted and a $\overline{\text{DTACK}}$ is received. When $\overline{\text{BERR}}$ is asserted and $\overline{\text{DTACK}}$ received, the exception taken depends upon the type of bus transaction being terminated. $\overline{\text{BERR}}$ is asserted and deasserted by the slave device a setup time prior to the rising edge of CLK34.
43	$\overline{\text{HOLD}}$	I	**Hold (Active-Low).** $\overline{\text{HOLD}}$ is asserted to suspend any further I/O transactions by the *Hobbit* microprocessor. $\overline{\text{HOLD}}$ is asserted or deasserted a setup time prior to the rising edge of CLK23. After $\overline{\text{HOLD}}$ is deasserted, bus transactions are allowed to start when the *Hobbit* microprocessor obtains ownership of the bus as $\overline{\text{HOLD}}$ is orthogonal to bus arbitration. In systems with slow 3-stating devices, assertion of $\overline{\text{HOLD}}$ may be necessary to allow the device time to relinquish the bus after $\overline{\text{DTACK}}$.
59	$\overline{\text{DTRI}}$	I	**Data 3-State (Active-Low).** The assertion of $\overline{\text{DTRI}}$ causes the asynchronous 3-stating of the data bus.
40	$\overline{\text{RETRY}}$	I	**Retry (Active-Low).** $\overline{\text{RETRY}}$ is asserted to retry the current bus transaction. $\overline{\text{RETRY}}$ is asserted or deasserted a setup time prior to the rising edge of CLK23. When $\overline{\text{RETRY}}$ is asserted during a valid bus transaction, the *Hobbit* microprocessor aborts the current bus transfer and masks the $\overline{\text{DTACK}}$ input. After $\overline{\text{RETRY}}$ is deasserted, the bus transaction is rerun after the *Hobbit* microprocessor obtains ownership of the bus as $\overline{\text{RETRY}}$ is orthogonal to bus arbitration. In systems with gateways through which two buses communicate with each other, the retry feature is required to break deadlock conditions when the two buses have simultaneous requests for their respective bus.
32	$\overline{\text{START}}$	O(3)	**Start Cycle (Active-Low).** Start cycle strobe is asserted by the current master to indicate the start of a bus transaction. $\overline{\text{START}}$ is asserted for only one clock cycle at the beginning of each bus transaction.
44	$\overline{\text{DTACK}}$	I	**Data Transfer Acknowledge (Active-Low).** During a normal bus transfer, this signal is used to terminate the transaction (data latched during read transaction, withdrawn during write transaction). $\overline{\text{DTACK}}$ is asserted and deasserted by the slave device a setup time prior to the rising edge of CLK34.

* I = input; O = output; P = power; (3) = 3-state; (R) = on-chip pull-up or pull-down resistor.

Pin Information (continued)

Pins Grouped Functionally (continued)

Table 1. Pin Descriptions (continued)

Pin	Symbol	Type*	Name/Description
77	HA2	O(3)	**Address Bus—Bit 2.**
71	HA3		**Address Bus—Bit 3.**
11	HA4		**Address Bus—Bit 4.**
5	HA5		**Address Bus—Bit 5.**
129	HA6		**Address Bus—Bit 6.**
123	HA7		**Address Bus—Bit 7.**
114	HA8		**Address Bus—Bit 8.**
108	HA9		**Address Bus—Bit 9.**
102	HA10		**Address Bus—Bit 10.**
95	HA11		**Address Bus—Bit 11.**
89	HA12		**Address Bus—Bit 12.**
79	HA13		**Address Bus—Bit 13.**
73	HA14		**Address Bus—Bit 14.**
67	HA15		**Address Bus—Bit 15.**
7	HA16		**Address Bus—Bit 16.**
1	HA17		**Address Bus—Bit 17.**
127	HA18		**Address Bus—Bit 18.**
105	HA19		**Address Bus—Bit 19.**
98	HA20		**Address Bus—Bit 20.**
92	HA21		**Address Bus—Bit 21.**
63	HA22		**Address Bus—Bit 22.**
16	HA23		**Address Bus—Bit 23.**
13	HA24		**Address Bus—Bit 24.**
121	HA25		**Address Bus—Bit 25.**
118	HA26		**Address Bus—Bit 26.**
116	HA27		**Address Bus—Bit 27.**
111	HA28		**Address Bus—Bit 28.**
86	HA29		**Address Bus—Bit 29.**
84	HA30		**Address Bus—Bit 30.**
82	HA31		**Address Bus—Bit 31.**
			The 30-bit address bus indicates word-aligned physical addresses. The byte enable signals, $\overline{BE[3:0]}$, are used for subword accesses.

* I = input; O = output; P = power; (3) = 3-state; (R) = on-chip pull-up or pull-down resistor.

Pin Information (continued)

Pins Grouped Functionally (continued)

Table 1. Pin Descriptions (continued)

Pin	Symbol	Type*	Name/Description
3	HD0	I/O(3)	**Data Bus—Bit 0.**
131	HD1		**Data Bus—Bit 1.**
128	HD2		**Data Bus—Bit 2.**
69	HD3		**Data Bus—Bit 3.**
65	HD4		**Data Bus—Bit 4.**
106	HD5		**Data Bus—Bit 5.**
104	HD6		**Data Bus—Bit 6.**
100	HD7		**Data Bus—Bit 7.**
17	HD8		**Data Bus—Bit 8.**
15	HD9		**Data Bus—Bit 9.**
12	HD10		**Data Bus—Bit 10.**
9	HD11		**Data Bus—Bit 11.**
6	HD12		**Data Bus—Bit 12.**
125	HD13		**Data Bus—Bit 13.**
122	HD14		**Data Bus—Bit 14.**
119	HD15		**Data Bus—Bit 15.**
117	HD16		**Data Bus—Bit 16.**
115	HD17		**Data Bus—Bit 17.**
112	HD18		**Data Bus—Bit 18.**
110	HD19		**Data Bus—Bit 19.**
96	HD20		**Data Bus—Bit 20.**
94	HD21		**Data Bus—Bit 21.**
90	HD22		**Data Bus—Bit 22.**
88	HD23		**Data Bus—Bit 23.**
85	HD24		**Data Bus—Bit 24.**
83	HD25		**Data Bus—Bit 25.**
81	HD26		**Data Bus—Bit 26.**

* I = input; O = output; P = power; (3) = 3-state; (R) = on-chip pull-up or pull-down resistor.

Pin Information (continued)

Pins Grouped Functionally (continued)

Table 1. Pin Descriptions (continued)

Pin	Symbol	Type*	Name/Description
78	HD27	I/O(3)	**Data Bus—Bit 27.**
75	HD28		**Data Bus—Bit 28.**
72	HD29		**Data Bus—Bit 29.**
62	HD30		**Data Bus—Bit 30.**
61	HD31		**Data Bus—Bit 31.** The 32-bit data bus conveys data to and from the *Hobbit* microprocessor. On byte writes, the active byte is indicated by the $\overline{BE[3:0]}$ signals with that byte replicated on the other inactive bytes. On half-word writes, the active half-word is indicated by the $\overline{BE[3:0]}$ signals with that half-word replicated on the inactive half-word. Looping back of the data bus is supported. After completion of a read transaction, if the current bus master retains ownership of the bus and there are no other transactions pending, the data just read by the *Hobbit* microprocessor is looped back onto the data bus to eliminate current leakage on the data bus.
20	$\overline{RD}$	O(3)	**Read (Active-Low).** When asserted (low), $\overline{RD}$ indicates a data read. When deasserted (high), it indicates a data write. It is asserted at the beginning of each bus transfer and is valid for the entire length of the transaction.
30	$\overline{LOCK}$	O(3)	**Bus Lock (Active-Low).** Multiple transfer bus lock. This signal is asserted to identify interlocked operations. The instruction set allows the *Hobbit* microprocessor to run interlocked operations for communication and message passing in a multiprocessor system. Also, the MMU asserts $\overline{LOCK}$ during misprocessing. Interlocked transfers in the *Hobbit* microprocessor are read-modify-write (RMW) cycles, although MMU misprocessing may abort the interlocked operation before the write starts. $\overline{LOCK}$ remains asserted through the write access. When the *Hobbit* microprocessor begins an interlocked operation, loss of bus ownership must not occur until $\overline{LOCK}$ is deasserted. Effectively, a dead cycle is inserted after a RMW operation. Interlocked transfers are not interruptible. Interlocked transfers may not be retried after the read completes. It is up to the system to enforce this restriction. If $\overline{RETRY}$ is asserted any time during an interlocked transfer, the retry is honored. It is illegal to assert $\overline{RETRY}$ after the read portion of the transfer since it causes the *Hobbit* microprocessor to abort the operation and become susceptible to bus arbitration, thus breaking the lock on the bus. Bus error can be asserted in either the read or write portion of the interlocked transfer. The interlocked operation is faulted with the appropriate exception sequence executed. If the operands being read by the interlocked instruction are in the stack cache, the lock signal is not asserted.

* I = input; O = output; P = power; (3) = 3-state; (R) = on-chip pull-up or pull-down resistor.

Pin Information (continued)

Pins Grouped Functionally (continued)

Table 1. Pin Descriptions (continued)

Pin	Symbol	Type*	Name/Description
19	$\overline{\text{NCACHE}}$	O(3)	**Not Cache (Active-Low).** The $\overline{\text{NCACHE}}$ output is provided for use with external caches to indicate that an address cannot be cached. When the PSW virtual/physical addressing mode bit is 1, the MMU uses the $\overline{\text{NCACHE}}$ output to indicate the status of the cache bit in various page table entries. When the PSW virtual/physical addressing mode bit is 0, $\overline{\text{NCACHE}}$ is asserted. When $\overline{\text{NCACHE}}$ is asserted, data should not be cached.
29 28	IOC0 IOC1	O(3)	**I/O Count—Bit 0.** **I/O Count—Bit 1.** These signals indicate the number of words remaining to be transferred. IOC[1:0] is used to determine the size of a block transfer being performed by the *Hobbit* microprocessor. These block transfers look like a series of bus transfers with $\overline{\text{START}}$ asserted for each and the new address provided by incrementing the lower address bits for each word transfer.
18	D/$\overline{\text{T}}$	O(3)	**Data/Text (Active-Low).** Indicates whether data or text is being accessed. It is asserted (high for data, low for text) at the beginning of each bus transfer and is valid for the entire length of the transaction.
22 23 24 26	$\overline{\text{BE0}}$ $\overline{\text{BE1}}$ $\overline{\text{BE2}}$ $\overline{\text{BE3}}$	O(3)	**Byte Enable 0 (Active-Low).** **Byte Enable 1 (Active-Low).** **Byte Enable 2 (Active-Low).** **Byte Enable 3 (Active-Low).** $\overline{\text{BE[3:0]}}$ indicate which bytes are valid during a data transfer, which may be either a read or a write. The bus supports 8-, 16-, 24-, or 32-bit data transfers (although the instruction set uses only 8-, 16-, or 32-bit data transfers). Combinations of the byte enable strobes are used to accomplish the desired word or subword transfer. Either little-endian or big-endian byte encoding may be selected for data via the PSW user little-endian bit or the CONFIG kernel little-endian bit for the user or kernel, respectively. Text is always big-endian encoding. **Byte Enable Strobe Encoding** **Pin — Bits Active** $\overline{\text{BE0}}$ — HD[7:0] $\overline{\text{BE1}}$ — HD[15:8] $\overline{\text{BE2}}$ — HD[23:16] $\overline{\text{BE3}}$ — HD[31:24]

* I = input; O = output; P = power; (3) = 3-state; (R) = on-chip pull-up or pull-down resistor.

Pin Information (continued)

Pins Grouped Functionally (continued)

Table 1. Pin Descriptions (continued)

Pin	Symbol	Type*	Name/Description
48 46 45	IL0 IL1 IL2	I	**Interrupt Request—Bit 0.** **Interrupt Request—Bit 1.** **Interrupt Request—Bit 2.** The *Hobbit* microprocessor recognizes six levels of interrupts encoded onto these lines. These signals should be supplied by an 8 to 3 priority encoder. See Table 21 on page 28 for the interrupt level encoding. When a valid interrupt is recognized, the *Hobbit* microprocessor requests ownership of the bus if it is not bus master. After becoming the bus master, the *Hobbit* microprocessor services the interrupt after aborting or completing the current instruction, depending on the type of instruction being executed. The internal latching of the interrupt is not predictable; the interrupting device must maintain its interrupt assertion until it is serviced. An external interrupt controller is required to resolve conflicts between simultaneously occurring interrupts.
52	TCK	I(R)	**Test Clock.** An externally gated clock signal with a 50% duty cycle. The changes on the TAP input signals (TMS and TDI) are clocked into the TAP controller, instruction register, or selected test data register on the rising edge of TCK. Changes at the TAP output signal (TDO) occur on the falling edge of TCK. This signal does not conform to IEEE 1149.1/D5 requirement of TCK being a free-running clock at all times. TCK must be stopped at 1 when internal BIT features are accessed. The TCK input has a built-in pull-up resistor to ensure that a high signal value is seen on an unterminated input.
49	TDI	I(R)	**Test Data Input.** TDI is clocked into the selected register data or instruction on the rising edge of TCK. The TDI input has a built-in pull-up resistor to ensure that a high signal value is seen on an unterminated input.
50	TMS	I(R)	**Test Mode Select.** TMS is a serial control input that is clocked into the TAP controller on the rising edge of TCK. The TMS input has a built-in pull-up resistor to ensure that a high signal value is seen on an unterminated input.
51	$\overline{\text{TRST}}$	I(R)	**Test Reset Input (Active-Low).** $\overline{\text{TRST}}$ is the reset input to the TAP controller. Assertion of this input forces the TAP controller into the reset state. The $\overline{\text{TRST}}$ input does not conform to IEEE 1149.1/D5 since it has a built-in pull-down resistor to ensure that a low signal value is seen on an unterminated input to force the TAP controller into the reset state.
38	TDO	O(3)	**Test Data Output.** The contents of the selected register data or instruction are shifted out of the TDO on the falling edge of TCK. TDO is 3-stated except when scanning of data is in progress.

* I = input; O = output; P = power; (3) = 3-state; (R) = on-chip pull-up or pull-down resistor.

Pin Information (continued)

Pins Grouped Functionally (continued)

Table 1. Pin Descriptions (continued)

Pin	Symbol	Type*	Name/Description
4, 10, 21, 27, 33, 37, 54, 60, 66, 70, 76, 87, 93, 99, 103, 109, 120, 126, 132	V_{DD}	P	**+3 V to +5 V Supply.** There are 19 V_{DD} pins.
2, 8, 14, 25, 31, 35, 41, 47, 58, 64, 68, 74, 80, 91, 97, 101, 107, 113, 124, 130	Vss	P	**Ground.** There are 20 Vss pins.

* I = input; O = output; P = power; (3) = 3-state; (R) = on-chip pull-up or pull-down resistor.

Pins Grouped Numerically

Table 2. Pin Descriptions

Pin	Symbol	Type*	Name
1	HA17	O(3)	**Address Bus—Bit 17.**
2	Vss	P	**Ground.**
3	HD0	I/O(3)	**Data Bus—Bit 0.**
4	V_{DD}	P	**+3 V to +5 V Supply.**
5	HA5	O(3)	**Address Bus—Bit 5.**
6	HD12	I/O(3)	**Data Bus—Bit 12.**
7	HA16	O(3)	**Address Bus—Bit 16.**
8	Vss	P	**Ground.**
9	HD11	I/O(3)	**Data Bus—Bit 11.**
10	V_{DD}	P	**+3 V to +5 V Supply.**
11	HA4	O(3)	**Address Bus—Bit 4.**
12	HD10	I/O(3)	**Data Bus—Bit 10.**
13	HA24	O(3)	**Address Bus—Bit 24.**
14	Vss	P	**Ground.**
15	HD9	I/O(3)	**Data Bus—Bit 9.**
16	HA23	O(3)	**Address Bus—Bit 23.**
17	HD8	I/O(3)	**Data Bus—Bit 8.**
18	D/$\overline{T}$	O(3)	**Data/Text (Active-Low).**
19	$\overline{NCACHE}$	O(3)	**Not Cache (Active-Low).**
20	$\overline{RD}$	O(3)	**Read (Active-Low).**
21	V_{DD}	P	**+3 V to +5 V Supply.**

* I = input; O = output; P = power; (3) = 3-state; (R) = on-chip pull-up or pull-down resistor.

Pin Information (continued)

Pins Grouped Numerically (continued)

Table 2. Pin Descriptions (continued)

Pin	Symbol	Type*	Name
22	$\overline{BE0}$	O(3)	Byte Enable 0 (Active-Low).
23	$\overline{BE1}$	O(3)	Byte Enable 1 (Active-Low).
24	$\overline{BE2}$	O(3)	Byte Enable 2 (Active-Low).
25	Vss	P	Ground.
26	$\overline{BE3}$	O(3)	Byte Enable 3 (Active-Low).
27	VDD	P	+3 V to +5 V Supply.
28	IOC1	O(3)	I/O Count—Bit 1.
29	IOC0	O(3)	I/O Count—Bit 0.
30	$\overline{LOCK}$	O(3)	Bus Lock (Active-Low).
31	Vss	P	Ground.
32	$\overline{START}$	O(3)	Start Cycle (Active-Low).
33	VDD	P	+3 V to +5 V Supply.
34	$\overline{BGACK}$	O	Bus Grant Acknowledge (Active-Low).
35	Vss	P	Ground.
36	$\overline{BREQ}$	O	Bus Request (Active-Low).
37	VDD	P	+3 V to +5 V Supply.
38	TDO	O(3)	Test Data Output.
39	$\overline{BGRANT}$	I	Bus Grant (Active-Low).
40	$\overline{RETRY}$	I	Retry (Active-Low).
41	Vss	P	Ground.
42	$\overline{BERR}$	I	Bus Error (Active-Low).
43	$\overline{HOLD}$	I	Hold (Active-Low).
44	$\overline{DTACK}$	I	Data Transfer Acknowledge (Active-Low).
45	IL2	I	Interrupt Request—Bit 2.
46	IL1	I	Interrupt Request—Bit 1.
47	VSS	P	Ground.
48	IL0	I	Interrupt Request—Bit 0.
49	TDI	I(R)	Test Data Input.
50	TMS	I(R)	Test Mode Select.
51	$\overline{TRST}$	I(R)	Test Reset Input (Active-Low).
52	TCK	I(R)	Test Clock.
53	$\overline{HRESET}$	I	Reset Signal (Active-Low).
54	VDD	P	+3 V to +5 V Supply.
55	CLK34	I	Phase 3 4 Clock.
56	$\overline{STOP}$	I	Stop Clocks (Active-Low)
57	CLK23	I	Phase 2 3 Clock.
58	Vss	P	Ground.
59	$\overline{DTRI}$	I	Data 3-State (Active-Low).

* I = input; O = output; P = power; (3) = 3-state; (R) = on-chip pull-up or pull-down resistor.

Pin Information (continued)

Pins Grouped Numerically (continued)

Table 2. Pin Descriptions (continued)

Pin	Symbol	Type*	Name
60	VDD	P	+3 V to +5 V Supply.
61	HD31	I/O(3)	Data Bus—Bit 31.
62	HD30	I/O(3)	Data Bus—Bit 30.
63	HA22	O(3)	Address Bus—Bit 22.
64	VSS	P	Ground.
65	HD4	I/O(3)	Data Bus—Bit 4.
66	VDD	P	+3 V to +5 V Supply.
67	HA15	O(3)	Address Bus—Bit 15.
68	VSS	P	Ground.
69	HD3	I/O(3)	Data Bus—Bit 3.
70	VDD	P	+3 V to +5 V Supply.
71	HA3	O(3)	Address Bus—Bit 3.
72	HD29	I/O(3)	Data Bus—Bit 29.
73	HA14	O(3)	Address Bus—Bit 14.
74	VSS	P	Ground.
75	HD28	I/O(3)	Data Bus—Bit 28.
76	VDD	P	+3 V to +5 V Supply.
77	HA2	O(3)	Address Bus—Bit 2.
78	HD27	I/O(3)	Data Bus—Bit 27.
79	HA13	O(3)	Address Bus—Bit 13.
80	VSS	P	Ground.
81	HD26	I/O(3)	Data Bus—Bit 26.
82	HA31	O(3)	Address Bus—Bit 31.
83	HD25	I/O(3)	Data Bus—Bit 25.
84	HA30	O(3)	Address Bus—Bit 30.
85	HD24	I/O(3)	Data Bus—Bit 24.
86	HA29	O(3)	Address Bus—Bit 29.
87	VDD	P	+3 V to +5 V Supply.
88	HD23	I/O(3)	Data Bus—Bit 23.
89	HA12	O(3)	Address Bus—Bit 12
90	HD22	I/O(3)	Data Bus—Bit 22.
91	VSS	P	Ground.
92	HA21	O(3)	Address Bus—Bit 21.
93	VDD	P	+3 V to +5 V Supply.
94	HD21	I/O(3)	Data Bus—Bit 21.
95	HA11	O(3)	Address Bus—Bit 11
96	HD20	I/O(3)	Data Bus—Bit 20.

* I = input; O = output; P = power; (3) = 3-state; (R) = on-chip pull-up or pull-down resistor.

Pin Information (continued)

Pins Grouped Numerically (continued)

Table 2. Pin Descriptions (continued)

Pin	Symbol	Type*	Name
97	VSS	P	Ground.
98	HA20	O(3)	Address Bus—Bit 20.
99	VDD	P	+3 V to +5 V Supply.
100	HD7	I/O(3)	Data Bus—Bit 7.
101	VSS	P	Ground.
102	HA10	O(3)	Address Bus—Bit 10
103	VDD	P	+3 V to +5 V Supply.
104	HD6	I/O(3)	Data Bus—Bit 6.
105	HA19	O(3)	Address Bus—Bit 19.
106	HD5	I/O(3)	Data Bus—Bit 5.
107	VSS	P	Ground.
108	HA9	O(3)	Address Bus—Bit 9.
109	VDD	P	+3 V to +5 V Supply.
110	HD19	I/O(3)	Data Bus—Bit 19.
111	HA28	O(3)	Address Bus—Bit 28.
112	HD18	I/O(3)	Data Bus—Bit 18.
113	VSS	P	Ground.
114	HA8	O(3)	Address Bus—Bit 8.
115	HD17	I/O(3)	Data Bus—Bit 17.
116	HA27	O(3)	Address Bus—Bit 27.
117	HD16	I/O(3)	Data Bus—Bit 16.
118	HA26	O(3)	Address Bus—Bit 26.
119	HD15	I/O(3)	Data Bus—Bit 15.
120	VDD	P	+3 V to +5 V Supply.
121	HA25	O(3)	Address Bus—Bit 25.
122	HD14	I/O(3)	Data Bus—Bit 14.
123	HA7	O(3)	Address Bus—Bit 7.
124	VSS	P	Ground.
125	HD13	I/O(3)	Data Bus—Bit 13.
126	VDD	P	+3 V to +5 V Supply.
127	HA18	O(3)	Address Bus—Bit 18.
128	HD2	I/O(3)	Data Bus—Bit 2.
129	HA6	O(3)	Address Bus—Bit 6.
130	VSS	P	Ground.
131	HD1	I/O(3)	Data Bus—Bit 1.
132	VDD	P	+3 V to +5 V Supply.

* I = input; O = output; P = power; (3) = 3-state; (R) = on-chip pull-up or pull-down resistor.

Instruction Set

Data Types

Six integer data types are supported: signed and unsigned bytes (8 bits), signed and unsigned half-words (16 bits), and signed and unsigned words (32 bits). Nonword operands are properly aligned and then expanded to 32 bits through sign extension (if signed) or clearing high-order bits (if unsigned).

After alignment and expansion, the 32-bit ALU performs the requested function. Carry and overflow are determined relative to the 32-bit result.

For destinations less than 32-bits, the least significant bits of the 32-bit ALU result are selected. Changing a value by truncation constitutes neither overflow nor carry.

True three-operand (triadic) instructions are not provided, but instruction encodings which provide two source operands and store the full 32-bit result in the accumulator are provided (see the ATT92010 Stacks section on page 30). This type of instruction is referred to as a two-and-a-half-operand instruction. For example, the mnemonic for an addition instruction of this type is given as ADD3 where a two-operand (dyadic) add is ADD. For this instruction, the two source operands are added and the full 32-bit result is stored in the accumulator.

Operand Addressing Modes

There are seven addressing modes:

- Immediate
- Absolute
- Stack offset
- Stack offset indirect
- Absolute indirect
- Program counter relative
- Register

The arithmetic logic unit (ALU) operations generally permit any of the first four of these addressing modes to be used with either operand. The valid addressing modes for each instruction are indicated in the detailed instruction descriptions in the *ATT92010* Hobbit *Microprocessor* Programmer's Guide. Any mode which is not explicitly mentioned for a given instruction should not be used. The sections below briefly describe each mode.

The operand can also have a suffix. The suffixes indicate the size of data operands. A missing suffix implies signed word operands.

- :B signed byte
- :UB unsigned byte
- :H signed half-word
- :UH unsigned half-word
- :W word

Immediate

In the immediate addressing mode, the operand value is stored in the instruction. Values up to 32 bits in length are permitted. Shorter values are appropriately sign or 0 extended before use.

Assembler language syntax: $data

Instruction Set (continued)

Operand Addressing Modes (continued)

Absolute

In the absolute addressing mode, the address of the operand is stored in the instruction.

Assembler language syntax: *$addr:suffix

Stack Offset

In the stack offset addressing mode, a signed, two's complement offset stored in the instruction (except for CATCH and ENTER, see the ATT92010 Stacks section on page 30) is added to the current stack pointer (CSP) value to obtain the operand address[1].

Assembler language syntax: Roffset:suffix

Stack Offset Indirect

In the stack offset indirect addressing mode, an offset is added to the CSP value to obtain the address of the operand's address. The offset must be word aligned[2].

Assembler language syntax: *Roffset:suffix

Absolute Indirect

In the absolute indirect addressing mode, the address of the operand's address is stored in the instruction. This mode is only used for the JMP (conditional jump instructions excluded), CALL, and LDRAA instructions, so that the operand value should be an instruction address which must be parcel (half-word) aligned.

Assembler language syntax: **$addr

Program Counter Relative

In the program counter relative addressing mode, a signed, two's complement offset stored in the instruction is added to the address of the instruction to obtain the operand value. This mode is only used for the JMP, CALL, and LDRAA instructions.

Assembler language syntax: label

Register

In the register addressing mode, the instruction in question is preceded by a CPU instruction.

A CPU instruction is never directly executed, but it serves to modify the next instruction's addressing modes for both operands. Code 0x7 allows access to the internal registers for use as data. The register number is specified in the operand (source/destination field). Only bits 3:0 are considered for determining the register number.

The upper bits are ignored but should be 0 for compatibility with future versions of the *Hobbit* microprocessor.

At most, one register may be read per instruction. If register 0x0 or 0xD through 0xF is specified, an unimplemented register exception sequence, exception ID 0x6, is performed. Registers can be read in user mode, but if there is a register write in user mode, a privilege violation exception sequence, exception ID 0x5, is performed.

Assembler language syntax: %REGISTER

Integer Arithmetic

The ATT92010 *Hobbit* microprocessor offers seven arithmetic instructions:

- ADD a, b ;add a to b
- DIV a, b ;divide b by a, signed
- MUL a, b ;multiply b by a
- REM a, b ;calculate the remainder of signed division of b by a
- SUB a, b ;subtract a from b
- UDIV a, b ;divide b by a, unsigned
- UREM a, b ;calculate the remainder of unsigned division of b by a

REM and UREM are defined in terms of DIV and UDIV, respectively. Operands a and b may be referenced using a variety of addressing modes, with sign interpretation given for byte and half-word arguments.

For the above instructions, the result is stored in b. ADD, DIV, MUL, REM, and SUB as well as other instructions also have a 2 1/2 address version (denoted by a trailing 3) where the result is stored in the accumulator (R4).

1. For negative offsets, off-chip stack accesses are performed and cache coherency is not maintained.
2. An alignment fault, 0x4, is executed if the offset is not word aligned.

Instruction Set (continued)

Integer Arithmetic (continued)

The Carry Bit

The PSW carry bit indicates the occurrence of a borrow during unsigned subtraction or of overflow during unsigned addition or multiplication. Unsigned overflow arises when a result exceeds unsigned(0xFFFFFFFF). In terms of the operations above, the PSW carry bit is set when:

Unsigned(b) – Unsigned(a) < 0

or unsigned overflow on an addition or multiplication:

Unsigned(b) {+ or *} Unsigned(a) > Unsigned(0xFFFFFFFF)

Unsigned overflow cannot occur in UDIV and UREM.

In the ADD operation, the adder computes the sum of a and b; the word result is delivered and, if carry-out occurs, the PSW carry bit is set. In the SUB operation, the two's complement of a is added to b, and the PSW carry bit is set only if no carry-out occurs.

The Overflow Bit

Analogous to the PSW carry bit, the PSW overflow bit signals the occurrence of signed overflow of the word result of an arithmetic operation; this is a result outside the interval:

[Signed(0x80000000) to Signed(0x7FFFFFFF)]

In terms of the operations above, the PSW overflow bit is set unless:

Signed(0x80000000) ≤ (Signed(b) {+, –, or *} Signed(a)) ≤ Signed(0x7FFFFFFF)

Signed overflow cannot occur in REM. Signed overflow does arise in DIV in precisely the case of 0x80000000 divided by –1 (i.e., 0xFFFFFFFF).

Division and Remainder

Unsigned overflow does not apply to UDIV because its dividend is at most unsigned(0xFFFFFFFF) and its divisor is no less than 1 (except for a zero divisor, which triggers a divide-by-zero exception), so its result is no greater than its dividend. A similar argument applies to DIV, except for the case of overflow.

Like UDIV, unsigned overflow does not apply to UREM. UD and UR are the word results of the UDIV and UREM operations, respectively. Apply these results to operands a and b. UDIV and UREM are related by the formula:

b = (UD*a) + UR, where 0 ≤ UR < a

with all values unsigned. UR is no greater than a and therefore no greater than unsigned(0xFFFFFFFF); hence, overflow cannot occur. A similar argument applies to REM.

Tagged Integer Arithmetic

- TADD a, b ;tagged add a into b
- TSUB a, b ;tagged subtract a from b

The tagged instructions ensure that the low 2 bits, called tags, of both operands are 0, before the arithmetic operation is performed. If either of the tags is nonzero, the PSW flag bit is set to 1, and the result is not stored. If both tags are zero, the result is stored only if the operation doesn't result in an arithmetic overflow. If the arithmetic overflow occurs, the PSW flag bit is set to 1 and the result is not stored. These instructions are useful in object-oriented languages where a given variable may represent different data types at different times during program execution.

Instruction Set (continued)

Fast Calling Sequence

The steps required for a function call are straightforward. Outgoing arguments are moved (or calculated) onto the stack frame (see the ATT92010 Stacks on page 30). In the event of word arguments, the first argument is stored at current stack pointer (CSP) + 4, the second at CSP + 8, etc. The CALL instruction performs an atomic move and jump operation, saving the return point at the CSP and loading the program counter (PC) with the address of the first instruction of the called function. The first instruction of the called function is usually ENTER which adjusts the CSP to allocate its new stack frame. The last instruction of the called function, RETURN, readjusts the CSP to deallocate its stack frame and then branches to the address pointed to by the CSP. Customarily, a CATCH follows the RETURN in user mode or when the user stack is enabled to refill the stack cache.

This function call overhead—call, allocate, deallocate, and return—can be as little as four clock cycles.

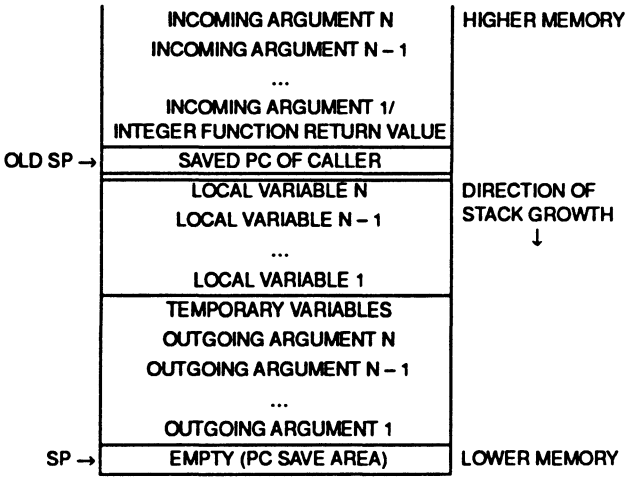


Figure 3. Typical Stack Frame (from the called functions point of view)

The stack grows downward in memory with the SP always pointing to the top of the stack. This free slot is where the PC is stored on a function call (or unimplemented instruction exception). This avoids having to adjust the CSP to save or restore the PC. The PC is the only machine register implicitly saved during a function call. Above the saved PC slot in the stack frame is an area large enough to store outgoing arguments for any call from the current function. Above the outgoing arguments are stored temporary values and local variables. Thus, outgoing arguments may be calculated in place with stack offset addressing modes. This statically allocated stack-frame allows the CSP to be updated only on function entry and function return. Traditional PUSH or POP[1] instructions which automatically adjust the CSP are intentionally avoided. Therefore, side effects to the CSP are nearly eliminated and operand address generation for subsequent instructions may smoothly proceed in a pipelined implementation.

Conditional Branches

Conditional branches are specified by first setting the PSW flag bit using CMPEQ, CMPGT, CMPHI, TESTC, or TESTV and then using a conditional jump (JMPTY, JMPTN, JMPFY, and JMPFN).

The jump doesn't need to be the next instruction after the flag is set. The pipeline runs more efficiently if three instructions that don't reference off-chip memory are between them.

The Y or N at the end of the conditional jump instruction is the prediction of the branch that will be taken (Y-jump, N-continue).

1. POPN is provided to deallocate from the stack frame and is useful in tail recursion.

Instruction Set (continued)

Instruction Format

Instructions are composed of 2 byte parcels and are encoded in one-, three-, and five-parcel lengths. The general instruction is encoded in five parcels, which allows for the encoding of two complete 32-bit addresses in each instruction. In general, the one- and three-parcel instructions are more compact encodings of five-parcel instructions. Instructions may have at most two operands, which, in general, have four addressing modes. For the dyadic instructions, one source doubles as destination or the accumulator is selected to serve as an implicit destination. The instruction formats are as follows:

- One-parcel formats (for zero-, one-, and two-operand instructions)
- Three-parcel formats (for one- and two-operand instructions)
- Five-parcel format (for two-operand instructions)

One-Parcel Formats

Many of the most common zero-, one-, and two-operand instruction types may be encoded in one parcel:

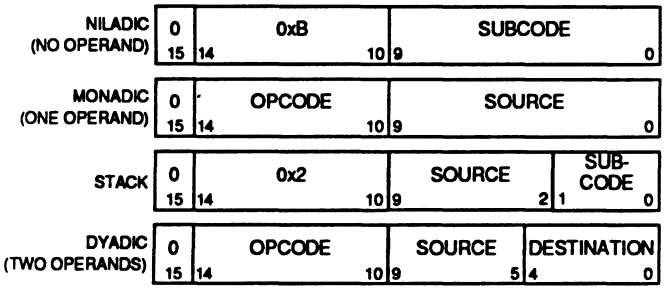


Figure 4. One-Parcel Instruction Formats

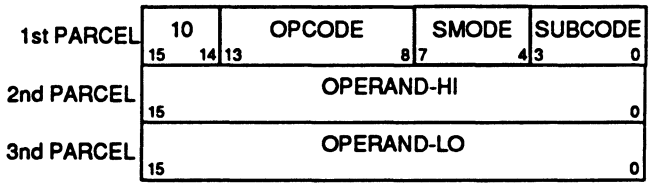
A 0 in the most significant bit distinguishes all one-parcel instruction formats. The subcode field distinguishes among the different niladic and stack instructions.

For operands, 5-bit immediate fields are sign extended, while 5-bit stack offset fields are zero extended. All 10-bit fields are zero extended except for CALL and JMP which are sign extended. The 8-bit fields are zero extended, except for ENTER, which is 1 filled.

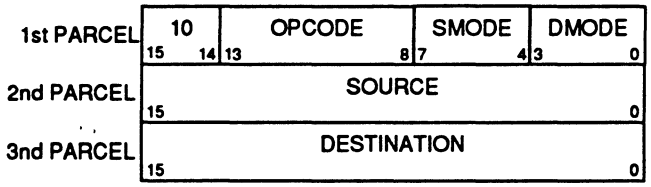
Note that operand alignment restrictions allow some address offsets to be scaled, thus extending the effective addressing range. The scaling of certain immediate constants is made possible by the specific operand value restrictions of the corresponding instructions. Five-bit offset values are multiplied by four before they are added to the SP. The 10-bit PC-relative offsets in JMP and CALL instructions are multiplied by 2 before they are used; the other 10-bit values are multiplied by four before they are used.

Three-Parcel Formats

Three-parcel instructions are distinguished by a 10 in the two most significant bits. The subcode field distinguishes among the different monadic instructions. The notation operand-lo refers to the low-order 16 bits, and operand-hi refers to the high-order 16 bits. A similar convention applies to the source and destination operands of the five-parcel dyadic instructions.



A. Monadic (One Operand)



B. Dyadic (Two Operand)

Figure 5. Three-Parcel Instruction Formats

The 16-bit source and destination fields are sign extended to 32 bits when they are used in immediate or offset modes. When the 16-bit source and destination fields are used as absolute addresses, extension of the upper 16 bits depends on the setting of the CONFIG PC extension bit. If the CONFIG PC extension bit is 1, bits 28:16 are replaced with 0 and bits 31:29 (the high-order 3 bits) are copied from bits 31:29 of the program counter. If the CONFIG PC extension bit is 0, the upper 16 bits are set to 0. The source and destination addressing mode fields are encoded in the same way for both three- and five-parcel instructions.

Instruction Set (continued)

Instruction Format (continued)

Five-Parcel Format

Five-parcel instructions are distinguished by a 11 in the two most significant bits. Five-parcel instructions are encoded similarly to three-parcel instructions.

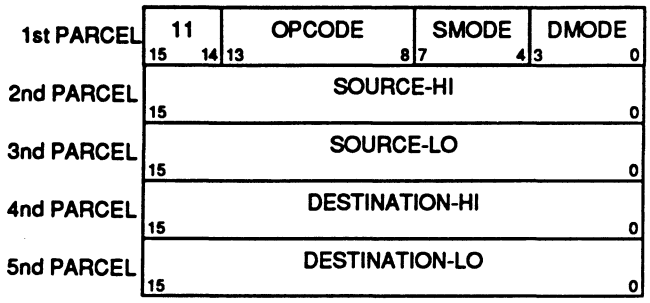


Figure 6. Five-Parcel Instruction Format

Instructions

The general instruction format is:

Instruction source, destination

where the instruction can contain a 3 indicating that the destination is the accumulator (R4). Otherwise, the destination is also the second operand.

The instructions can be divided into eight categories. The following special notations are used: [] and (|). ADD[3], for example, indicates that both ADD and ADD3 instructions exist. JMP (F|T)(Y|N) indicates that JMPFY, JMPFN, JMPTY, and JMPTN instructions exist.

Table 3. Arithmetic Instructions

Instruction	Function
ADD[3]	Add
ADDI	Add interlocked
DIV[3]	Divide
MUL[3]	Multiply
REM[3]	Remainder
SUB[3]	Subtract
UDIV	Unsigned divide
UREM	Unsigned remainder

Table 4. Logical Instructions

Instruction	Function
AND[3]	Bitwise logical and
ANDI	Bitwise logical and interlocked
OR[3]	Bitwise logical or
ORI	Bitwise logical or interlocked
XOR[3]	Bitwise logical exclusive or

Table 5. Shift Instructions

Instruction	Function
SHL[3]	Left shift
SHR[3]	Arithmetic right shift
USHR[3]	Logical right shift

Table 6. Compare Instructions

Instruction	Function
CMPEQ	Equality comparison
CMPGT	Signed greater than comparison
CMPHI	High comparison (unsigned greater than)

Table 7. Move Instructions

Instruction	Function
DQM	Double-word move (destination suffix :B) Quad-word move (destination suffix :W)
LDRAA	Load PC-relative address into the accumulator
MOV	Move
MOVA	Move address

Table 8. Tagged Instructions

Instruction	Function
TADD	Tagged addition
TSUB	Tagged subtraction

Instruction Set (continued)

Instructions (continued)

Table 9. Program Control Instructions

Instruction	Function
CALL	Call function
CATCH	Fill stack cache
CRET	Return from kernel with context
ENTER	Allocate stack space
JMP	Unconditional jump
JMP(F|T)(Y|N)	Conditional jump based on PSW flag bit
KCALL	Kernel call
KRET	Return from kernel
POPN	Free N entries from stack space
RETURN	Free stack space and return from function

Table 10. Other Instructions

Instruction	Function
CLRE	Clear PSW enter guard bit
CPU	Register mode addressing
FLUSHI	Flush the decoded instruction cache
FLUSHP	Flush the prefetch buffer cache
FLUSHPBE	Flush an entry in prefetch buffer cache
FLUSHPTE	Flush a page table entry in the TLBs or NPSRs
NOP	No operation
TESTC	Copy PSW carry bit to PSW flag bit and clear carry bit
TESTV	Copy PSW overflow bit to PSW flag bit and clear overflow bit

Instruction Set (continued)

Instruction (OPCODE/SUBCODE) Encodings

Table 11. One-Parcel Instruction Encodings, Monadics/Dyadics

opcode[4:3]	opcode[2:0]							
	000	001	010	011	100	101	110	111
00	KCALL	CALL	stack*	JMP	JMPFN	JMPFY	JMPTN	JMPTY
01	unimp†	unimp†	MOV.WS	niladic‡	unimp†	ADD3.WS	AND3.CS	AND.SS
10	CMPEQ.CS	CMPGT.SS	CMPGT.CS	CMPEQ.SS	ADD.CS	ADD3.CS	ADD.SS	ADD3.SS
11	MOV.SS	MOV.IS	MOV.SI	MOV.II	MOV.CS	MOVA.SS	SHL3.CS	SHR3.CS

Notes:
C = 5-bit immediate
I = 5-bit indirect stack offset
S = 5-bit stack offset
W = 5-bit word-aligned immediate
* See Table 12.
† The unimplemented instruction sequence is performed.
‡ See Table 13.

Table 12. One-Parcel Instruction Encodings, Stack

subcode[1:0]			
00	01	10	11
ENTER	CATCH	RETURN	POPN

Table 13. One-Parcel Instruction Encodings, Niladics

subcode[9:3]	subcode[2:0]							
	000	001	010	011	100	101	110	111
0000000	CPU	KRET	NOP	FLUSHI	FLUSHP	CRET	FLUSHD*	unimp*
0000001	TESTV	TESTC	CLRE	unimp*	unimp*	unimp*	unimp*	unimp*
000001x	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*
00001xx	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*
0001xxx	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*
001xxxx	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*
01xxxxx	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*
1xxxxxx	trap†	trap†	trap†	trap†	trap†	trap†	trap†	trap†

* The unimplemented instruction sequence is performed.
† The niladic trap through VB + 8 is performed.

Instruction Set (continued)

Instruction (OPCODE/SUBCODE) Encodings (continued)

Table 14. Three-Parcel Encodings

opcode[5:3]	opcode[2:0]							
	000	001	010	011	100	101	110	111
000	monadic†	ORI	ANDI	ADDI	MOVA	UREM	MOV	DQM
001	unimp*	unimp*	unimp*	unimp*	TADD	TSUB	unimp*	unimp*
010	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*
011	unimp*	unimp*	unimp*	unimp*	unimp*	CMPGT	CMPHI	CMPEQ
100	SUB	OR	AND	ADD	XOR	REM	MUL	DIV
101	unimp*	unimp*	unimp*	unimp*	SHR	USHR	SHL	UDIV
110	SUB3	OR3	AND3	ADD3	XOR3	REM3	MUL3	DIV3
111	unimp*	unimp*	unimp*	unimp*	SHR3	USHR3	SHL3	unimp*

* The unimplemented instruction sequence is performed.
† See Table 15.

Table 15. Three-Parcel Instruction Subcodings, Monadic

subcode[9:3]	subcode[2:0]							
	000	001	010	011	100	101	110	111
0	KCALL	CALL	RETURN	JMP	JMPFN	JMPFY	JMPTN	JMPTY
1	CATCH	ENTER	LDRAA	FLUSHPTE	FLUSHPBE	FLUSHDCE*	unimp*	POPN

* The unimplemented instruction sequence is performed.

Table 16. Five-Parcel Instruction Encodings

opcode[5:3]	opcode[2:0]							
	000	001	010	011	100	101	110	111
000	unimp*	ORI	ANDI	ADDI	MOVA	UREM	MOV	DQM
001	unimp*	unimp*	unimp*	unimp*	TADD	TSUB	unimp*	unimp*
010	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*	unimp*
011	unimp*	unimp*	unimp*	unimp*	unimp*	CMPGT	CMPHI	CMPEQ
100	SUB	OR	AND	ADD	XOR	REM	MUL	DIV
101	unimp*	unimp*	unimp*	unimp*	SHR	USHR	SHL	UDIV
110	SUB3	OR3	AND3	ADD3	XOR3	REM3	MUL3	DIV3
111	unimp*	unimp*	unimp*	unimp*	SHR3	USHR3	SHL3	unimp*

* The unimplemented instruction sequence is performed.

Instruction Set (continued)

Addressing Mode (SMODE/DMODE) Encodings

Table 17. General Addressing Mode Encodings

Mode	Code	Description
*$addr:B	0x0	Byte absolute
*$addr:UB	0x1	Unsigned byte absolute
*$addr:H	0x2	Half-word absolute
*$addr:UH	0x3	Unsigned half-word absolute
Roffset:B	0x4	Byte stack offset
Roffset:UB	0x5	Unsigned byte stack offset
Roffset:H	0x6	Half-word stack offset
Roffset:UH	0x7	Unsigned half-word stack offset
*Roffset:B	0x8	Byte stack offset indirect
*Roffset:UB	0x9	Unsigned byte stack offset indirect
*Roffset:H	0xA	Half-word stack offset indirect
*Roffset:UH	0xB	Unsigned half-word stack offset indirect
*$addr:W	0xC	Word absolute
Roffset:W	0xD	Word stack offset
*Roffset:W	0xE	Word stack offset indirect
$data	0xF	Immediate

Table 18. CPU Modified Addressing Mode Encodings

Mode	Code	Description
register	0x7	CPU prefixed
*$addr:W	0xC	Word absolute
Roffset:W	0xD	Word stack offset
*Roffset:W	0xE	Word stack offset indirect
$data	0xF	Immediate

Table 19. CALL/JMP Addressing Mode Encodings

Mode	Code	Description
**$addr	0xC	Absolute indirect
*Roffset	0xD	Stack offset indirect
Label	0xE	Program counter relative
*$addr	0xF	Absolute

Table 20. Source/Destination Register Encodings

Register	Code
MSP	0x1
ISP	0x2
SP	0x3
CONFIG	0x4
PSW	0x5
SHAD	0x6
VB	0x7
STB	0x8
FAULT	0x9
ID	0xA
TIMER1	0xB
TIMER2	0xC
unimp	0xD
unimp	0xE
unimp	0xF

Prefetching Strategy

The ATT92010 *Hobbit* microprocessor provides two types of instruction fetching selectable through the CONFIG prefetch mode bit: aggressive prefetching and demand fetching. When aggressive prefetching is enabled (CONFIG prefetch mode bit = 1), the prefetch unit on the microprocessor fetches text, which has not been previously fetched and stored in the prefetch buffer memory, in quad-word pieces consisting of two double-word I/O requests. Text is prefetched sequentially until a branch (predicted jump, unconditional jump, CALL, CRET, KCALL, KRET, or RETURN) is decoded. If the target of the branch is encoded in the instruction (nonindirect), prefetching then continues from the target (if it is not already in the prefetch buffer); if the target is indirect, prefetching stops and waits for a demand fetch request from the execution unit. A demand fetch is requested if the execution unit takes a mispredicted or indirect branch and the target has not been previously decoded. If at any time while the prefetch unit is prefetching sequential code and following predicted branches a demand fetch is requested, any I/O requested by the unit will complete, and prefetching begins anew from the execution unit requested target.

If demand fetching is enabled (CONFIG prefetch mode bit = 0), the prefetch unit only issues an I/O request for text when it is requested by the execution unit and is not found in the prefetch buffer. The I/O request is made for a double word, and all instructions contained in the double word and any subsequent instructions found in the prefetch buffer are decoded, but prefetching ceases until another demand fetch is requested by the execution unit. Demand fetching is the default mode after reset. Whether to use aggressive prefetching or demand prefetching depends on the application.

Static Branch Prediction and Branch Folding

Branches break the flow of instruction execution and can degrade the performance of a pipelined microprocessor. Furthermore, the target of a conditional jump is not known until the instruction is executed. The *Hobbit* microprocessor solves these problems in two ways. First, the instruction format provides a static branch prediction field which is set at compile time, indicating whether it is more likely for the conditional branch to be taken or not. Since the prefetch decode unit (PDU) continued prefetching along the predicted path of a conditional jump, the instructions can be issued and executed into the pipeline without any discontinuity. Second, the PDU assigns a next-PC and alternate-next-PC field for each decoded instruction. Thus, for selected single-parcel instructions or a three-parcel instruction, A, if the following instruction, B, is a jump, the next-PC field for the instruction A is the (predicted) target of the instruction B and in the case B is conditional alternate-next-PC is the nonpredicted target of B.

Instruction Tracing

Instruction tracing is supported by the PSW trace basic block or trace instruction bits. These bits control when tracing is enabled. If an instruction is traceable, a trace exception is taken after the instruction completes execution. The PC saved on the interrupt stack is the PC of the next instruction.

Instructions before folded branches cannot be traced (i.e., if a jump is folded into the previous instruction, the trace will occur after the jump). To circumvent this from occurring, all jumps must be encoded as three-parcel and, hence, will not be folded.

Event sequences are nontraceable. This includes exceptions and interrupts. The unimplemented instruction sequence is traceable if the trace bits are not altered. CRET, KCALL, and KRET are always nontraceable.

Event Processing

There are several sequences which can be triggered in the *Hobbit* microprocessor that are not usually invoked by the regular instruction set. These events include, in order of priority:

1. reset
2. interrupt
3. exception

The sequences executed by the *Hobbit* microprocessor for each of these events are listed in the following sections. In all cases, interrupts are inhibited while an event processing sequence (the sequence that initiates the event handler) is in progress.

As described in the following sections, the processing of exceptions and interrupts includes the saving of the PC and PSW on the interrupt stack. For instructions that change the PC, the current PC is described below.

- CALL and JUMP: If the location pointed to by the instruction cannot be referenced, a fetch-fault results and the PC stored on the interrupt stack is the target PC, not the PC of the instruction. If the indirection word of an indirect instruction cannot be referenced, a read-fault results and the PC stored on the interrupt stack is that of the instruction.
- KCALL: If the location pointed to by the KCALL PC entry in the vector cannot be referenced, a fetch-fault results and the PC stored on the interrupt stack is the target PC, not the PC of the original KCALL.
- CRET, KRET, and RETURN: If the location pointed to by the new PC value cannot be referenced, a fetch-fault results and the PC stored on the interrupt stack is the new PC value, not the address of the instruction.

Reset

The ATT92010 *Hobbit* microprocessor enters the reset sequence when:

- The external reset pin is asserted.
- A memory fault, which is signaled either externally or by the MMU,
  - —occurs when attempting to read or write the interrupt stack during any event processing sequence.
  - —occurs when attempting to read from the vector table during any event processing sequence.

The reset sequence is:

1. Disable interrupts
2. Flush the PFB and IC
3. if ($\overline{\text{HRESET}}$)
   SHAD = 0x0
   else
   SHAD = PSW
4. PSW = 0x0
5. CONFIG = 0x0
6. PC = 0x0
7. Enable NMI interrupts

After a reset, SHAD is set to either 0x0 or the current PSW depending upon which type of reset occurred. Independent of the type of reset, the PFB and IC are flushed and the PSW, CONFIG, and PC are initialized to 0x0. 0x0 in the PSW register sets the execution level to kernel, with physical addressing enabled, tracing disabled, interrupts inhibited, and the ISP as the CSP. 0x0 in the CONFIG register disables all on-chip caches, disables timer interrupts, and selects demand prefetching. 0x0 in the PC register starts executing instructions at physical address 0x0.

Note: If the reset sequence was initiated by $\overline{\text{HRESET}}$ being asserted, the SP and the MSP are undefined. The caches should not be enabled until these registers are assigned values since the range check circuitry would not know whether an address should access the on-chip stack cache or off-chip memory.

Event Processing (continued)

Interrupt

An interrupt is signaled when an external device requests service on the interrupt request input lines IL[2:0] or either timer1 or timer2 overflows with the respective interrupt enabled. The three input lines associated with external interrupts and the timer interrupts, which are asserted at level 1, are compared with the PSW interrupt priority level field and the CONFIG timer interrupt enables. If the interrupt request is less than the PSW interrupt priority level field, the interrupt can be serviced. A PSW interrupt priority level field of 7 allows interrupts at levels 6:0. A PSW interrupt priority level field of 0 inhibits interrupts 6:1 and allows only interrupts at level 0, which is referred to as a nonmaskable interrupt (NMI).

Table 21. Interrupt Levels

IL[2:0]	Interrupt Level
000	NMI
001	Level 1
010	Level 2
011	Level 3
100	Level 4
101	Level 5
110	Level 6
111	No interrupt

The interrupt request input lines IL[2:0] must be asserted with the same value for at least two cycles before an interrupt is recognized by the *Hobbit* microprocessor. The interrupt should remain asserted until the interrupt handler clears it. If the interrupt is accepted: the request enters at the top of the execution-unit pipeline. Then all further interrupts are disabled until completion of the interrupt sequence. The ATT92010 *Hobbit* microprocessor does not indicate when it is servicing an interrupt other than the I/O caused by the interrupt handler.

An NMI can be generated by setting IL[2:0] to 0x0. An interrupt at level 0 is edge sensitive in that when asserted, it must be deasserted for at least two cycles before another interrupt at any level is recognized. When any interrupt enters the execution pipeline, all interrupts are disabled, including NMI. After the interrupt sequence completes, if the NMI is still asserted, it will be serviced.

Most instructions complete execution before the interrupt request enters the top of the execution-unit pipeline. CATCH, ENTER, MUL[3], DIV[3], REM[3], UDIV, and UREM are interruptible. The CATCH portion of CRET is interruptible. The PC stored on the interrupt stack is the address of the interrupted instruction for transparently resuming execution. CATCH, ENTER, and the CATCH portion of CRET continue as opposed to restarting.

Interrupt Sequence

When the interrupt is serviced, the sequence is as follows:

1. Disable interrupts
2. if (CSP == ISP) ISP = SHAD
   else SP =SHAD
3. *(ISP–8) = PC of interrupted instruction
   /* Becomes R8 with respect to new ISP */
4. *(ISP–4) = PSW
   /* Becomes R12 with respect to new ISP */
5. ISP –= 16
6. SHAD = ISP
7. PC = *(VB + 16 + (4 x interrupt level))
8. PSW &= 0xFFFF0000
9. Enable NMI interrupts

where the interrupt level is the value of the IL[2:0] lines producing the interrupt. Note that the interrupt sequence is almost the same as the KCALL sequence. In particular, the event frame left on the interrupt stack is the same, so a KRET instruction is sufficient for returning from an interrupt; interrupts are disabled during this processing.

Event Processing (continued)

Exceptions

Exceptions signal an error in a program. The exceptions recognized by the ATT92010 *Hobbit* microprocessor are listed in Table 22.

Table 22. Exception Identifiers

Exception	Code
Integer zero-divide	0x1
Trace	0x2
Illegal instruction	0x3
Alignment fault	0x4
Privilege violation	0x5
Unimplemented register	0x6
Fetch fault	0x7
Data read fault	0x8
Data write fault	0x9
Memory access I/O bus fault	0xA
MMU table walk bus fault	0xB

The exception handler must always be present.

Exception Sequence

1. Disable interrupts
2. if (CSP == ISP) ISP = SHAD
   else SP = SHAD
3. * (ISP–12) = exception identifier
   /* Becomes R4 with respect to new ISP */
4. *(ISP–8) = PC of faulted instruction
   /* Becomes R8 with respect to new ISP */
5. *(ISP–4) = PSW
   /* Becomes R12 with respect to new ISP */
6. ISP –= 16
7. SHAD = ISP
8. PC = *(VB + 4)
9. PSW &= 0xFFFF0000
10. Enable NMI interrupts

The sequence is almost the same as that of KCALL. If the target address of a CALL, CRET, JMP, KCALL, KRET, or RETURN instruction, or of an interrupt, causes a memory fault, the PC saved on the interrupt stack is the target PC, not the address of the current instruction.

In the case of exception IDs 0x8 and 0x9, the 32-bit operand aligned virtual address of faulted access is saved in the fault register.

For a text fetch bus error or a data read bus error, the PC placed on the interrupt stack is the address of the instruction with the faulting address. For a data write bus error, the PC placed on the interrupt stack is not the PC of the instruction associated with the faulted access. Due to the unhinged nature of the stores in the *Hobbit* microprocessor, the PC stored is the PC of the instruction which was at the bottom of the execution pipeline when the fault occurred, and not the PC of the instruction with which the faulted store is associated.

Unimplemented Instruction

An attempt to execute an unimplemented opcode results in an unimplemented instruction sequence. This sequence is faster than the exception sequence facilitating software emulation of extended instructions. Since an unimplemented instruction can occur in either execution mode, the unimplemented instruction handler should be in both the user and kernel address space.

If an unimplemented instruction has an addressing mode which is illegal for that instruction class, it is considered an illegal instruction (exception ID 0x3). Specifically:

- An unimplemented monadic instruction is considered illegal if it has a nonword addressing mode (<0xC).
- An unimplemented instruction is considered illegal if it follows a CPU instruction and contains an illegal addressing mode, or combination of modes.
- RETURN with a negative operand.

There are no tests performed upon the addressing modes of unimplemented dyadic instructions which do not follow CPU instructions.

Unimplemented Instruction Sequence

1. *(CSP) = PC of unimplemented opcode
2. PC = *(VB + 12)

where CSP is either SP or ISP, depending upon the state of the PSW current stack pointer bit.

Event Processing (continued)

Trapped Niladics

An attempt to execute a one-parcel niladic with an opcode in the range 0x200 through 0x3FF results in a variant of the previously described unimplemented instruction sequence called the trapped niladic exception. This sequence is the same as the unimplemented instruction sequence except VB + 8 is used for the vector. The trapped niladic handler should be in both the user and kernel address space.

Trapped Niladic Sequence

1. *(CSP) = PC of unimplemented opcode
2. PC = *(VB + 8)

where CSP is either SP or ISP, depending upon the state of the PSW current stack pointer bit.

Event Processing Priority

The priorities assigned to each event type request are as follows:

1. Reset
2. Interrupts
3. Trace
4. Instruction fetch faults
5. Illegal instructions
6. Unimplemented instructions/trapped niladics
7. Unimplemented registers
8. Alignment faults
9. Data read and write and read bus error faults
10. Privilege violation
11. Divide by zero

Events 3 through 11 are associated with a particular instruction, while the higher-priority events (reset and interrupts) can occur independent of the instruction being executed. During some internal sequences, interrupts are disabled. Many events are mutually exclusive of each other and cannot occur at the same time or within the same instruction.

ATT92010 Stacks

The ATT92010 *Hobbit* microprocessor is equipped with two stacks, the interrupt stack and the user stack. The ISP register points to the interrupt stack, and the SP register points to the user stack. The current stack pointer (CSP) selects the current stack based on the PSW current stack pointer bit. When an interrupt or exception is serviced, the CSP automatically switches to the ISP.

The interrupt stack resides in memory that should be valid at all times. Part or all of the user stack resides in the stack cache if it is enabled by the CONFIG stack cache bit.

The Stack Cache

The goal of the stack cache is to keep the top elements of the stack in high-speed registers (the stack grows to lower addresses). The stack cache consists of a bank of 64 registers (4 bytes wide) organized as a circular buffer maintained by two registers: the maximum stack pointer (MSP) and the stack pointer (SP). Both the MSP and the SP are 28-bit registers holding quad-word addresses. The MSP contains the address above the highest address of the data that is currently kept in the stack cache registers; the SP delimits the lowest address of data in the stack cache. Therefore, only a simple range check is needed to determine if an address resides within the stack cache. If $SP \leq ADDR < MSP$, it falls within the stack cache. Although the stack cache limits are maintained on quad-word boundaries, the stack cache is byte addressable and appears as normal memory. All virtual addresses generated to access data may freely reference the stack cache.

Since, the stack cache can contain the top 64 words of the stack, most automatic variables and incoming and outgoing arguments will be in the stack cache. The stack cache is, therefore, a major factor in efficient instruction execution.

ATT92010 Stacks (continued)

Stack Cache Maintenance

Six instructions maintain the stack cache: CALL, CATCH, CRET, ENTER, POPN, and RETURN. CALL places the return address on the top of the stack and branches to the target address. CATCH guarantees that the stack cache is filled at least as deep as the number of the bytes specified in its operand and is used after a CALL instruction to ensure that an optimal portion of the stack is on-chip. CRET is used by the kernel to load a new SP and MSP and execute the function of CATCH to fill the stack cache. CRET also loads a new PSW and program counter address and is used for context switches. ENTER allocates space on the new stack frame by subtracting its operand, the size of the new stack frame, from the SP. POPN deallocates the current stack frame by adding its argument to the SP. RETURN deallocates the current stack frame by adding its argument to the SP and then branching to the return address on the top of the stack.

ENTER and CATCH are also used to handle the cases where the stack cache circular buffer is not large enough to accommodate the entire stack frame. When a new procedure is entered, the ENTER instruction attempts to allocate a new set of registers equal to the size of the new stack frame. If free register space exists in the circular buffer, then only the SP needs to be modified. If not, then the entries nearest the MSP are flushed back to main memory. Two cases exist:

- If the new stack frame size is less than 256 bytes, then only the stack frame size minus the number of free entries must be flushed.
- If the new stack frame size is ≥256 bytes, then all valid stack cache entries are flushed and only part of the new stack frame nearest the SP is kept in the stack cache.

A garbage collection function needs the PSW enter bit for proper operation. When 1, this bit indicates that the stack frame contains uninitialized data. After the data is initialized, software should clear this bit.

After a procedure returns to the caller, the number of stack cache entries that were flushed since the call is unknown, so some entries may need to be restored from off-chip memory. The argument of the CATCH instruction specifies the number of stack cache entries that must be valid before execution can continue efficiently. The CATCH argument is used as a stack offset, and a virtual address is generated. If this calculated address resides within the stack cache, execution continues. However, if it lies outside the address range of valid stack cache entries, quad-words pointed to by the MSP are restored from off-chip memory to the stack cache, and the MSP is incremented until either CATCH is satisfied or the stack cache is full. The CATCH instruction behaves much like an assertion, since usually no entries need to be restored and CATCH takes only one clock cycle.

Integer Accumulator

The integer accumulator is not a fixed hardware register. It is the word in memory above the word addressed by the CSP. The CSP is either the SP or the interrupt stack pointer (ISP), as discussed above. The integer accumulator normally resides on-chip in the stack cache, but it may be off-chip if the SP = MSP or CSP = ISP.

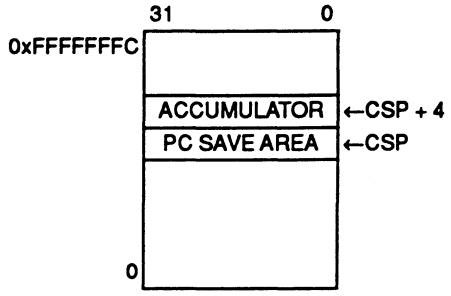


Figure 7. Integer Accumulator

Stack Precautions

The stack cache is conceptually a cache for memory. If an address is generated in any processing stage, e.g., indirect address calculations, the stack cache is referenced if that address is greater than or equal to the SP and less than the MSP. This conceptual model is violated when executing with CSP = ISP. There are no problems with memory accesses as long as the stack cache, based at the SP, and the interrupt stack, based at the ISP, do not overlap. For similar reasons, the following addresses must not lie between the SP and MSP:

- The vector table, defined by the vector base (VB)
- The address translation tables used by the MMU
- Any text address

Control Registers

Table 23. ATT92010 *Hobbit* Microprocessor Control Registers

Name	Description	Name	Description
CONFIG	Configuration Register	SHAD	Shadow Register
FAULT	Fault Register	SP	Stack Pointer
ID	Identification Register	STB	Segment Table Base
ISP	Interrupt Stack Pointer	TIMER1	Timer1 Register
MSP	Maximum Stack Pointer	TIMER2	Timer2 Register
PC	Program Counter	VB	Vector Base
PSW	Program Status Word		

CONFIG—Configuration Register

The configuration register (CONFIG) is set to 0x0 upon reset.

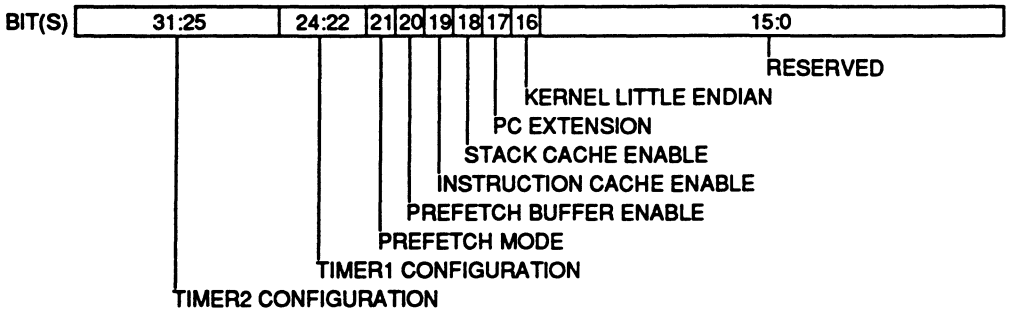


Figure 8. CONFIG—Configuration Register

Table 24. CONFIG—Configuration Register

Bit(s)	Description
31:25	**Timer2 Configuration.** A 7-bit field which configures timer2. —Bit 29:25 select the internal event which increments timer2. **Bit 29 Bit 28 Bit 27 Bit 26 Bit 25 Event** 0 0 0 0 0 Count clock cycles. 0 0 0 0 1 Count completed instructions (folded branches are not counted). 1 1 1 1 1 Do not increment the timer, a low-power feature. —Bit 30. If 0, timer2 is on all the time (with reference to bits 29:25). If 1, the timer only increments in kernel mode (PSW execution level bit is 0). —Bit 31. If 0, timer2 does not generate an interrupt. If 1, timer2 generates an interrupt using a timer2 vector when an overflow occurs (goes from 0xFFFFFFFF to 0x0). This is a level one interrupt. An external level one interrupt and a timer1 interrupt have priority over timer2.

Note: Special precautions must be taken when modifying the configuration register. The number of NOPs which must come after the register write varies according to which bits are being modified and the number of wait-states being used by I/O transactions. The preferred means of modifying CONFIG is to follow the CONFIG write by either a CRET or KRET.

Control Registers (continued)

CONFIG—Configuration Register (continued)

Table 24. CONFIG—Configuration Register (continued)

Bit(s)	Description
24:22	**Timer1 Configuration.** A 3-bit field which configures timer1. —Bit 22. If 0, timer1 counts clock cycles. If 1, timer1 counts completed instructions (folded branches are not counted). —Bit 23. If 0, timer1 is on all the time (with reference to bit 22). If 1, the timer only increments in kernel mode (PSW execution level bit is 0). —Bit 24. If 0, timer1 does not generate an interrupt. If 1, timer1 generates an interrupt using a timer1 vector when an overflow occurs (goes from 0xFFFFFFFF to 0x0). This is a level one interrupt. An external level one interrupt has priority over timer1.
21	**Prefetch Mode.** This bit controls prefetching of instructions. If 0, prefetching off-chip is not performed; predecoding from the prefetch buffer into the instruction cache is performed. If 1, aggressive prefetching is performed. See the Prefetching Strategy section on page 26 for more information.
20	**Prefetch Buffer Enable.** A 0 disables the prefetch buffer from hitting; a 1 enables it. The prefetch buffer is neither flushed nor altered when this bit is modified.
19	**Instruction Cache Enable.** A 0 disables the instruction cache from hitting; a 1 enables it. The instruction cache is neither flushed nor altered when this bit is modified.
18	**Stack Cache Enable.** A 0 disables the stack cache from hitting; a 1 enables it. The stack cache is neither flushed nor altered when this bit is modified.
17	**PC Extension.** A 0 selects 0 extension of 16-bit absolute addresses; a 1 selects the extension of 16-bit absolute addresses where bits 31:29 are copied from bits 31:29 of the PC and bits 28:16 are set to 0.
16	**Kernel Little Endian.** A 0 selects data as big endian in kernel mode; a 1 selects data as little endian in kernel mode.
15:0	**Reserved.** They return 0 when read and should be written with 0 on CONFIG writes.

Note: Special precautions must be taken when modifying the configuration register. The number of NOPs which must come after the register write varies according to which bits are being modified and the number of wait-states being used by I/O transactions. The preferred means of modifying CONFIG is to follow the CONFIG write by either a CRET or KRET.

FAULT—Fault Register

This register reports the 32-bit operand aligned virtual address for the processing of exception IDs 0x8 and 0x9.

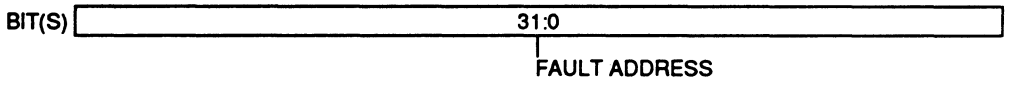


Figure 9. FAULT—Fault Register

Table 25. FAULT—Fault Register

Bit(s)	Description
31:0	**Fault Address.** This is the address causing the current exception for use by the exception handler.

Control Registers (continued)

ID—JTAG ID Register

This register is the JTAG device identification register and is readable by serial shifting through the test access port (TAP) and through normal register access. This register is read only. No operation is performed if this register is written to.

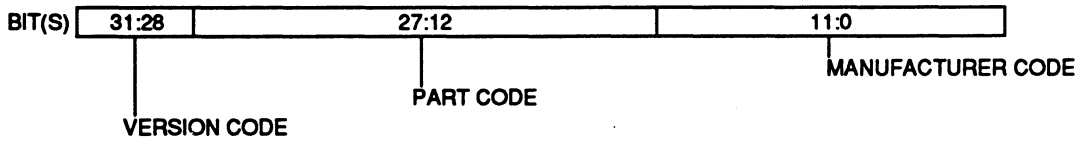


Figure 10. ID—Identification Register

Table 26. ID—Identification Register

Bit(s)	Description
31:28	**Version Code.** This field is 0x0 for mask one and 0x1 for mask two.
27:12	**Part Code.** This field is 0x0 for the ATT92010 *Hobbit* microprocessor.
11:0	**Manufacturer Code.** This field is 0x3B for AT&T Microelectronics.

ISP—Interrupt Stack Pointer

The interrupt stack pointer (ISP) is used to generate addresses (i.e., as the base address in stack offset modes, to locate the accumulator, and as the pointer manipulated by the instructions CALL, RETURN, POPN, and ENTER) whenever the PSW current stack pointer bit is 0. The ISP is not associated with the stack cache. The instructions CRET, KCALL, and KRET, and operating system sequences, interrupts, and exceptions use the ISP to maintain a stack of event blocks. The ISP must be valid at all times. A fault on any ISP based address during event processing results in the resetting of the ATT92010 *Hobbit* microprocessor. Address translation is performed if the MMU is enabled by setting the PSW virtual/physical addressing mode bit to 1.

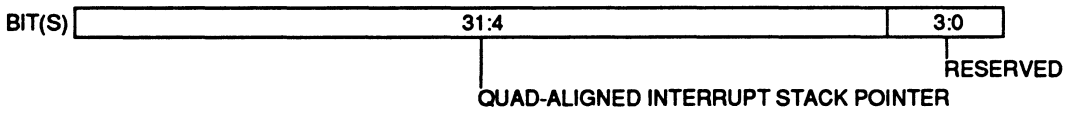


Figure 11. ISP—Interrupt Stack Pointer

Table 27. ISP—Interrupt Stack Pointer

Bit(s)	Description
31:4	**Quad-Aligned Interrupt Stack Pointer.** This is the address of the interrupt stack.
3:0	**Reserved.** These bits return 0 when read.

Control Registers (continued)

MSP—Maximum Stack Pointer

The maximum stack pointer (MSP), in conjunction with the SP, is associated with the on-chip stack cache. If the current stack pointer is the SP then, any address which is greater than or equal to the SP and less than the MSP hits in the stack cache.

Stack cache hits when SP ≤ address < MSP.

On a memory access which hits in the stack cache, data is fetched or stored in the cache, not in external memory. The MSP must be greater than or equal to the SP and less than or equal to SP + 256 (stack cache size), or the result of stack cache accesses are dependent upon context and, therefore, are unpredictable.

Whenever the SP is the direct destination of an instruction, through a CPU-prefixed instruction with the SP as the destination, the MSP is updated with the same value. This defines an empty stack cache (SP = MSP). The MSP is manipulated implicitly by CATCH, CRET, ENTER, POPN, and RETURN. Hence, the MSP should only be modified by stack manipulation instructions. Address translation is performed if the MMU is enabled by setting the PSW virtual/physical addressing mode bit to 1.

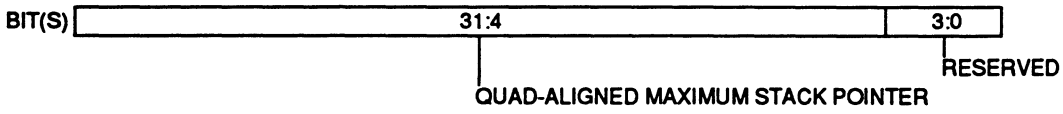


Figure 12. MSP—Maximum Stack Pointer

Table 28. MSP—Maximum Stack Pointer

Bit(s)	Description
31:4	**Quad-Aligned Maximum Stack Pointer.** This is the address above top of user stack.
3:0	**Reserved.** These bits return 0 when read.

PC—Program Counter

The program counter (PC) addresses the instruction which is currently being executed. Instructions are aligned on parcel (half-word) boundaries. Since parcels are composed of 2 bytes, the PC is always a multiple of two and the low-order bit is always 0. The PC cannot be directly manipulated by a general instruction. It can only be read or modified by control-flow instructions CALL, CRET, JMP, KCALL, KRET, and RETURN and read by the move instruction LDRAA.

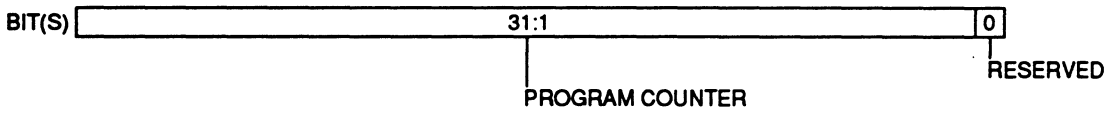


Figure 13. PC—Program Counter

Table 29. PC—Program Counter

Bit(s)	Description
31:1	**Program Counter.** This is the address of the current instruction.
0	**Reserved.**

Control Registers (continued)

PSW—Program Status Word

The program status word (PSW) is set to 0x0 upon reset.

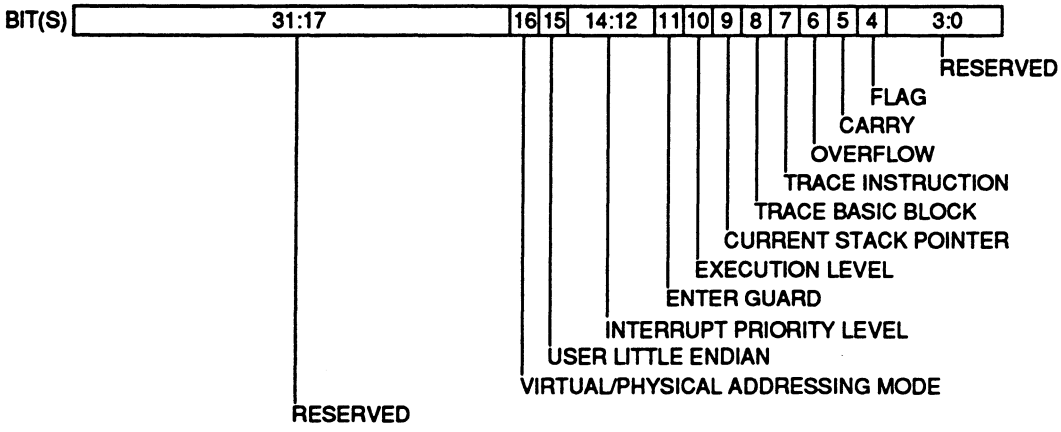


Figure 14. PSW—Program Status Word

Table 30. PSW—Program Status Word

Bit(s)	Description
31:17	**Reserved.**
16	**Virtual/Physical Addressing Mode.** If 0, physical addressing (memory management disabled) is enabled, and $\overline{\text{NCACHE}}$ is asserted. If 1, virtual addressing is enabled (memory management enabled). Special precautions must be taken when explicitly modifying this bit. If it is explicitly modified, the section of code executing must be mapped physical address = virtual address. The safest means of manipulating this bit is through KRET.
15	**User Little Endian.** If 0, data is selected as big endian in user mode. If 1, data is selected as little endian in user mode.
14:12	**Interrupt Priority Level.** Interrupts are accepted when the requesting device level (IL[2:0]) is less than interrupt priority level or equal to 0. When these bits equal 7, all interrupts are enabled.
11	**Enter Guard.** Set on an uneventful ENTER. This bit is not cleared when the PSW is read.
10	**Execution Level.** If 0, execution at the kernel level is performed. If 1, execution at the user level is performed.

Note: The exception and interrupt sequences only alter the lower 16 bits of the PSW. To remain restartable, the carry and overflow bits are not cleared on reading the PSW until the instruction completes. Reads of the PSW are not interlocked against flag setting. If an instruction sets the flag, carry, or overflow bits, there must be at least two intervening instructions, which do not use or modify these bits, before the PSW can be accurately read.

Control Registers (continued)

PSW—Program Status Word (continued)

Table 30. PSW—Program Status Word (continued)

Bit(s)	Description
9	**Current Stack Pointer.** If 0, the ISP is used as the CSP for stack operations. If 1, the SP is used as the CSP for stack operations. If this bit is modified by a direct write to the PSW, thereby changing the CSP, it is necessary to update SHAD to the value of the new SP. This update is handled automatically by CRET, KCALL, and KRET. If this bit is set to 1, and it was previously 0, the instruction modifying the PSW should be followed by the instruction MOV %SP,%SHAD. If this bit is set to 0 when it was previously 1, the next instruction should be MOV %ISP,%SHAD. Due to interrupts and exceptions, it is recommended that this bit not be modified by a direct write to the PSW since the above operations cannot be guaranteed to be atomic.
8	**Trace Basic Block.** Controls basic block tracing. If 1, the *Hobbit* microprocessor executes instructions until a CALL, RETURN, or any jump (folded or not) instruction, referred to as the N instruction, executes. The instruction following instruction N, referred to as N + 1, is not permitted into the execution unit, and a trace instruction is generated internally. This trace instruction blocks the pipeline and forces the *Hobbit* microprocessor to take a trace exception using the PC of the N + 1 instruction as the exception PC. As branch folding is performed prior to the trace identifier, folded branches are not explicitly traceable. If both the trace instruction and the trace basic block bits are set to 1, the function is that of the trace instruction.
7	**Trace Instruction.** Controls instruction tracing. When 1, the *Hobbit* microprocessor allows the next instruction, N, to execute normally. The instruction following instruction N, referred to as N + 1, is not permitted into the execution unit, and a trace instruction is generated on the fly. This trace instruction blocks the pipeline and forces the *Hobbit* microprocessor to take a trace exception using the PC of the N + 1 instruction as the exception PC. As branch folding is performed prior to the trace identifier, folded branches are not explicitly traceable. If both the trace instruction and the trace basic block bits are set to 1, the function is that of the trace instruction.
6	**Overflow.** If 0, this bit indicates that an operation did not generate a signed overflow. If 1, this bit indicates that an operation generated a signed overflow. This bit is not cleared by a read of the PSW.
5	**Carry.** If 0, this bit indicates that an operation did not generate an unsigned overflow. If 1, this bit indicates that an operation generated an unsigned overflow. This bit is not cleared by a read of the PSW.
4	**Flag.** Set/cleared by CMP, TADD, TESTC, TESTV, and TSUB instructions. This bit is not cleared by a read of the PSW.
3:0	**Reserved.** These bits are reserved. They return 0 when read and must be written with 0 on PSW writes.

Note: The exception and interrupt sequences only alter the lower 16 bits of the PSW. To remain restartable, the carry and overflow bits are not cleared on reading the PSW until the instruction completes. Reads of the PSW are not interlocked against flag setting. If an instruction sets the flag, carry, or overflow bits, there must be at least two intervening instructions, which do not use or modify these bits, before the PSW can be accurately read.

Control Registers (continued)

SHAD—Shadow Register

The shadow register (SHAD) is a copy of the CSP. It is maintained by the ATT92010 *Hobbit* microprocessor's internal sequences to facilitate restarting of instructions. In the course of CRET, ENTER, KCALL, KRET, and RETURN instructions, or any time the CSP is modified, SHAD is automatically updated to be consistent with the CSP.

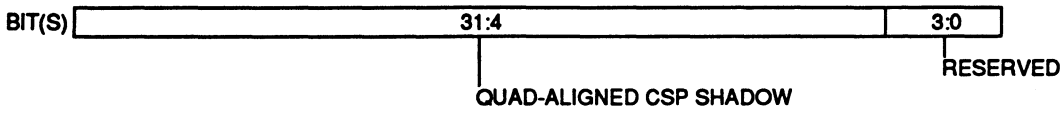


Figure 15. SHAD—Shadow Register

Table 31. SHAD—Shadow Register

Bit(s)	Description
31:4	**Quad-Aligned CSP Shadow.** These bits contain a copy of the CSP.
3:0	**Reserved.** These bits return 0 when read.

Note: If the PSW current stack pointer bit is modified by a direct write to the PSW, thereby changing the CSP, it is necessary to update SHAD to the value of the new SP. KCALL and KRET handle this automatically.

SP—Stack Pointer

The stack pointer (SP) addresses the top of the stack. The stack grows downwards toward memory location zero. The SP is used to generate addresses (i.e., as the base address in offset modes, to locate the accumulator, and as the pointer manipulated by CALL, ENTER, POPN, and RETURN) whenever the PSW current stack pointer bit is 1. Address translation is performed if the MMU is enabled by setting the PSW virtual/physical addressing mode bit to 1.

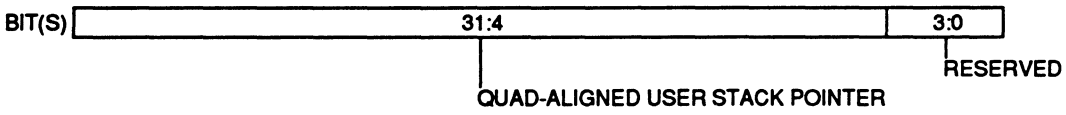


Figure 16. SP—Stack Pointer

Table 32. SP—Stack Pointer

Bit(s)	Description
31:4	**Quad-Aligned User Stack Pointer.** This is the address of the user stack.
3:0	**Reserved.** These bits return 0 when read.

Control Registers (continued)

STB—Segment Table Base

This register contains a pointer to the start of the segment table used in address translation when virtual addressing is turned on by the PSW virtual/physical addressing mode bit. The base of the segment table is always page-size aligned, 4 Kbyte boundary. The STB is only used during misprocessing, to fill entries in the on-chip TLB or segment registers. When the STB is written, the TLBs and segment registers of the MMU are flushed, invalidating all entries. Neither the physically addressed PFB, the virtually addressed IC, nor the virtually addressed SC are flushed. Cache coherency is the responsibility of the user.

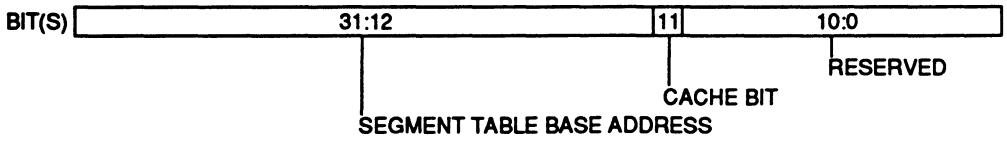


Figure 17. STB—Segment Table Base

Table 33. STB—Segment Table Base

Bit(s)	Description
31:12	**Segment Table Base Address.** This is the page-aligned base address of the segment table.
11	**Cache Bit.** A cacheable bit that is copied to the cacheable pin whenever a segment table access is made during misprocessing, indicating if segment table entries should be cached. If 1, $\overline{\text{NCACHE}}$ is deasserted and caching of segment table entries is allowed.
10:0	**Reserved.** Return 0 when read.

Control Registers (continued)

TIMER1—Timer1 Register

This register can be configured by the CONFIG timer1 configuration bits to count various events.

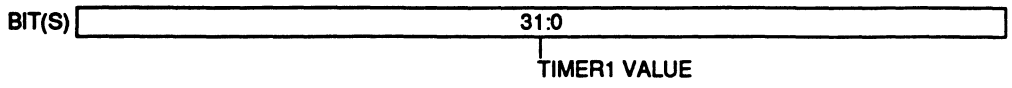


Figure 18. TIMER1—Timer1 Register

Table 34. TIMER1—Timer1 Register

Bit(s)	Description
31:0	**Timer1 Value.** These bits contain the count value for Timer1.

TIMER2—Timer2 Register

This register can be configured by the CONFIG timer2 configuration bits to count various events.

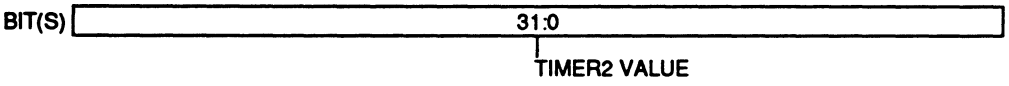


Figure 19. TIMER2—Timer2 Register

Table 35. TIMER2—Timer2 Register

Bit(s)	Description
31:0	**Timer2 Value.** These bits contain the count value for Timer2.

Control Registers (continued)

VB—Vector Base

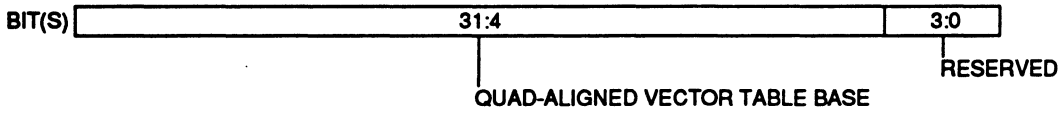


Figure 20. VB—Vector Base

Table 36. VB—Vector Base

Bit(s)	Description
31:4	**Quad-Aligned Vector Table Base.** These bits are used as the base of a table which contains transfer addresses used by KCALL, interrupts, and exceptions. Address translation is performed if the MMU is enabled by setting the PSW virtual/physical addressing mode bit to 1. The vector table, as shown below, should always be available. If an access to the vector table entry is faulted, the *Hobbit* microprocessor resets. The exception PC handler should be present in memory, since a memory fault would cause an infinite loop until the interrupt stack is exhausted and the *Hobbit* microprocessor resets. Additionally, the niladic trap and unimplemented instruction handlers must be in the user memory space since these handlers can be accessed while in user mode. VB + 52→ FP EXCEPTION VB + 48→ TIMER2 INTERRUPT VB + 44→ TIMER1 INTERRUPT VB + 40→ INTERRUPT 6 VB + 36→ INTERRUPT 5 VB + 32→ INTERRUPT 4 VB + 28→ INTERRUPT 3 VB + 24→ INTERRUPT 2 VB + 20→ INTERRUPT 1 VB + 16→ NONMASKABLE INTERRUPT VB + 12→ UNIMPLEMENTED INSTRUCTION VB + 8→ NILADIC TRAPS VB + 4→ EXCEPTION PC VB → KCALL PC
3:0	**Reserved.** These bits return 0 when read.

Addressing and Alignment Restrictions

The numbering of bits within bytes and words corresponds to that in the *DEC VAX*[1], *Intel*[2] 80X86, and *Motorola*[3] 680X0. The numbering of bytes within data words is selectable independently for the user mode by the PSW user little-endian bit and the kernel mode by the CONFIG kernel little-endian bit, respectively. When the PSW user little-endian bit and CONFIG kernel little-endian bit equals 0, the numbering of bytes within data words corresponds to that in the *IBM*[4] 370 and *Motorola* 680X0 in the user/kernel mode (see Figure 21.)

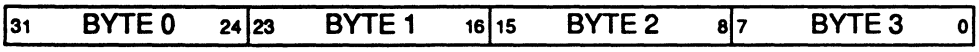


Figure 21. Big-Endian Byte Ordering

When the PSW user little-endian bit and CONFIG kernel little-endian bit equals 1, the numbering of bytes within data words corresponds to that in the *VAX* and *Intel* 80X86 in the user/kernel mode (see Figure 22.)

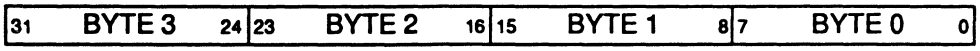


Figure 22. Little-Endian Byte Ordering

Text is always in big-endian order. The ATT92010 *Hobbit* microprocessor fetches only words; bytes and half-words are accessed by extracting them from the surrounding word. During reads, the byte enables indicate which bytes are to be extracted from within the word being fetched. All writes are done to word addresses, with the appropriate byte enables asserted.

All operand addresses should be naturally aligned for the operand type[5]. If an operand fetch or operand store is to an address which is not properly aligned for the data type, an alignment exception is signaled. Instructions must be aligned on half-word boundaries, although no exception is signaled. Alignment occurs as the least significant bit of the address is ignored for text fetches.

Memory Management

The *Hobbit* microprocessor has an on-chip memory management unit (MMU), which translates virtual addresses (if enabled), as seen by a programmer, into physical addresses. Two methods for address translation are provided: paged and nonpaged segments (see Figure 23 and Figure 24, respectively.)

The 32-bit virtual address space is divided into 1,024 segments, each representing 4 Mbytes of virtual addresses with a 4 Mbyte alignment. Paged segments are further divided into 4 Kbyte pages. Nonpaged segments provide a variable-sized (4 Kbyte to 4 Mbyte in 4 Kbyte increments) contiguous segment of memory. In paged segment address translation, each page can be mapped anywhere in the 32-bit physical address space on a 4 Kbyte boundary.

Address translation is enabled by setting the PSW virtual/physical bit to 1. To speed paged segment address translation, the *Hobbit* microprocessor has two TLBs, one for text addresses and one for data addresses. Each TLB has 32 entries and is fully associative. Two nonpaged segment registers (NPSRs), one for a text address and one for a data address, speed nonpaged segment address translation.

1. *DEC* and *VAX* are trademarks of Digital Equipment Corporation.
2. *Intel* is a trademark of Intel Corporation.
3. *Motorola* is a registered trademark of Motorola, Inc.
4. *IBM* is a registered trademark of International Business Machines Corporation.
5. Byte on byte, half-words on half-word, words on word address boundaries.

Memory Management (continued)

Additionally, to provide a zero-cycle program counter translation, a micro-TLB is provided for text references in the present page. This micro-TLB contains the last translation used by the prefetch unit and provides zero-cycle address translation. If the micro-TLB misses, one cycle is required for update if the address translation hits in the text TLB or text NPSR. If an address is not contained in the appropriate TLB or NPSR, the on-chip MMU automatically fetches the appropriate entry by walking the memory management tables.

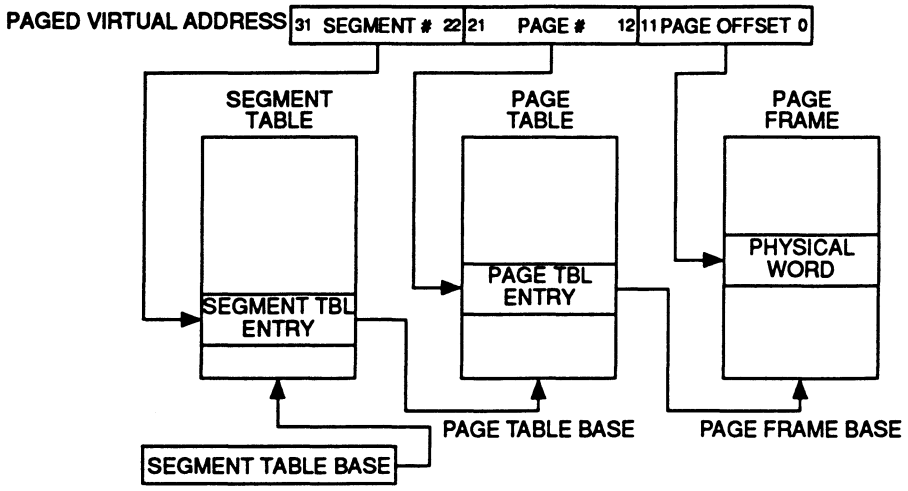


Figure 23. Paged Segment Address Mapping

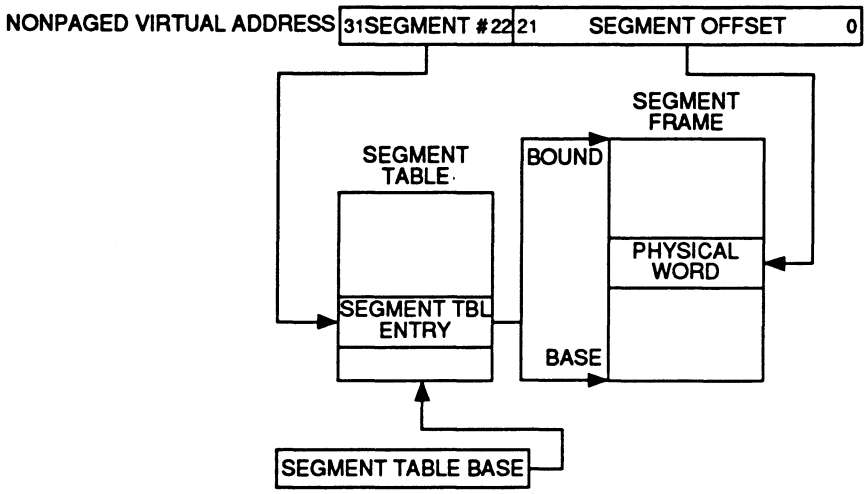


Figure 24. Nonpaged Segment Address Mapping

Memory Management (continued)

Virtual Address Mapping

All addresses in the *Hobbit* microprocessor are translated by walking a series of map tables (see Figure 23 and Figure 24 on page 43). All map tables in the memory mapping scheme are 4,096 bytes long (one page frame). All addresses contained within a memory management table are physical addresses, so address translation is not recursive. Address mapping checks the validity of virtual addresses and translates them into physical addresses. A virtual address is flagged as illegal if one of the following happens:

- There is no valid physical mapping.
- User execution level code attempts to access kernel execution level addresses.
- A store is attempted to read-only data.

Any violation is signaled as a memory fault as described below:

Fetch fault: If, during an address translation for text, there is no physical mapping or an attempt is made to access a kernel only page while in user mode, this fault is signaled. Note that a fetch fault is generated only on demand fetches and only stops fetching, until a demand fetch, if aggressive fetching is enabled by the PSW prefetch bit.

Read fault: If, during an address translation for reading data, there is no physical mapping or an attempt is made to access a kernel only page while in user mode, this fault is signaled. This fault can be ignored if the read was requested because of a mispredicted branch.

Write fault: If, during an address translation for either writing data or while executing one of the stack manipulation instructions, there is no physical page, an attempt is made to access a kernel only page in user mode, or an attempt is made to write to a nonwritable page, this fault is signaled.

Paged Segment Addresses

A page frame is a contiguous region of 4,096 bytes, beginning at an address evenly divisible by 4,096 (the low 12-bits of the address are all 0). Because all page frames begin on page boundaries, additions are not necessary to calculate addresses. When paged segment translation is in use, virtual addresses are divided into the following three fields (see Figure 23 on page 43):

- Segment number
- Page number
- Page offset

Nonpaged Segment Addresses

When nonpaged segment translation is in use, virtual addresses are divided into the following two fields (see Figure 24 on page 43):

- Segment number
- Segment offset

Memory Management (continued)

Segment Tables

The segment number selects one entry from 1,024 entries in the segment table—a 4 Kbyte table located in one page frame in physical memory. Each segment table entry is 4 bytes long and contains the base address of a page table or the base address and size of a nonpaged segment. The base address of the segment table is contained in the segment table base register (STB).

The address of a segment table entry is formed by concatenating the upper 20-bits of the segment table base register with the upper 10 bits of the virtual address: the base address field in the segment table base defines the beginning of a segment table in physical memory, and the segment number field of the virtual address defines a word within the segment table.

There are two possible formats for a segment table entry. Paged segments have referenced and modified bits for enhanced memory management. Nonpaged segments only require the segment table to resolve references.

Paged Segment Table Entries

The segment table for paged segments defines 1,024 segments each 1,024 pages long (for a total of 4,294,967,296 bytes). Segments are defined as a series of pages, so there may be holes in a segment's address space. There is no length specification for a segment: the validity of constituent pages defines a segment's extent. Each paged segment table entry defines a page table.

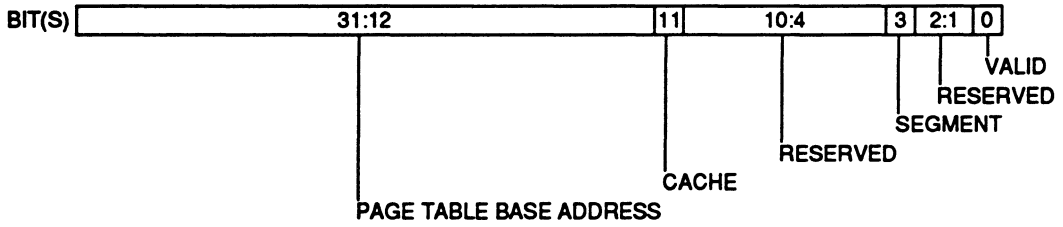


Figure 25. Paged Segment Table Entry

Table 37. Paged Segment Table Entry

Bit(s)	Name/Description
31:12	**Page Table Base Address.** The base address in physical memory of the page table.
11	**Cache.** If 1, $\overline{\text{NCACHE}}$ is deasserted when fetching page table entries.
10:4	**Reserved.**
3	**Segment.** 0 for paged segment translation.
2:1	**Reserved.**
0	**Valid.** If 1, the entry is valid.

Memory Management (continued)

Segment Tables (continued)

Nonpaged Segment Table Entries

The segment table for nonpaged segments defines 1,024 segments each from 4,096 bytes to 4 Mbytes long.

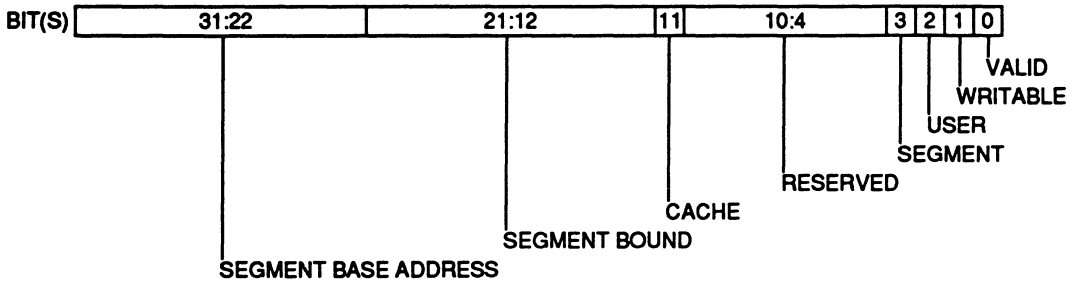


Figure 26. Nonpaged Segment Table Entry

Table 38. Nonpaged Segment Table Entry

Bit(s)	Name/Description
31:22	**Segment Base Address.** These bits contain the base address of the segment in physical memory.
21:12	**Segment Bound.** These bits contain the size of the segment, ranging from 4,096 bytes (0x0) to 4 Mbytes (0x3FF) in increments of 4,096 bytes.
11	**Cache.** If 0, $\overline{\text{NCACHE}}$ is asserted when accessing this segment. Text fetches will not be cached in the prefetch buffer cache, but they will be cached in the decoded instruction cache. If 1, $\overline{\text{NCACHE}}$ is deasserted when accessing nonpaged segments. This bit has no effect on the use of the stack cache.
10:4	**Reserved.**
3	**Segment.** A 1 for nonpaged segment translation.
2	**User.** If 1, the segment can be accessed at user execution level (all valid segments can be accessed at kernel level).
1	**Writable.** If 1, the segment can be written (all valid segments can be read).
0	**Valid.** If 1, the segment is valid.

The segment offset field of the virtual address defines the byte within the segment frame in which the virtual address is mapped. The physical address consists of the segment base address from the segment table entry concatenated with the segment offset field of the virtual address. If a protection violation is detected, no memory access is made and a memory fault exception is executed.

Mixed Paged and Nonpaged Segment Tables

Since the segment bit in the segment table entry controls if the segment table entry is paged or nonpaged, a segment table can contain both paged and nonpaged entries.

Memory Management (continued)

Page Tables

The address of a page table entry is formed by concatenating the upper 20-bits of the segment table entry with bits 21:12 of the virtual address (which is the page number).

A page table entry defines the physical address corresponding to the virtual address and provides protection information and other data available for paging algorithms. The reference and modified bits are automatically set by the on-chip MMU, but they must be cleared by software when needed.

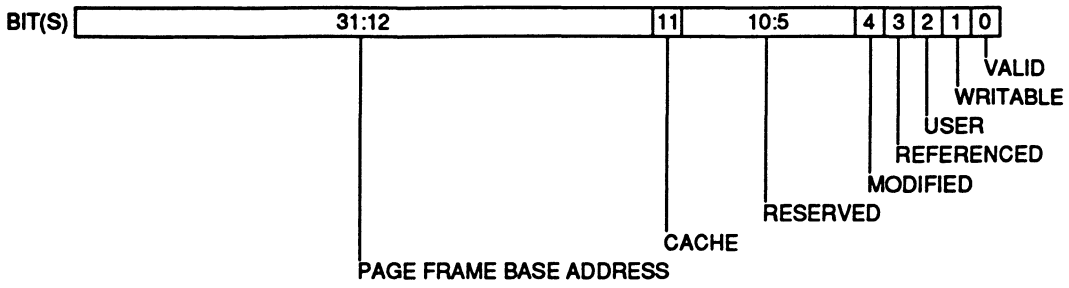


Figure 27. Page Table Entry

Table 39. Page Table Entry

Bit(s)	Name/Description
31:12	**Page Frame Base Address.** These bits contain the base address in physical memory of the page frame.
11	**Cache.** If 0, $\overline{\text{NCACHE}}$ is asserted when accessing this page. Text fetches will not be cached in the prefetch buffer cache, but they will be cached in the decoded instruction cache. If 1, $\overline{\text{NCACHE}}$ is deasserted when accessing this page. This bit has no effect on the use of the stack cache.
10:5	**Reserved.**
4	**Modified.** Set to 1 when a write occurs within the page. On subsequent writes to this page, the memory copy of the page table entry is not accessed to set this bit again. If a direct write to the memory copy of the page table entry changes this bit, the entry should be flushed from the TLB using the FLUSHPTE instruction.
3	**Referenced.** Set to 1 when a page is first referenced. On subsequent references to this page, the memory copy of the PTE is not accessed to set this bit again. If a direct write to the memory copy of the PTE changes this bit, the entry should be flushed from the TLB using the FLUSHPTE instruction.
2	**User Bit.** If 1, the page can be accessed at user execution level (all valid pages can be accessed by the kernel).
1	**Writable.** If 1, the page can be written (all valid pages can be read).
0	**Valid.** If 1, the page is valid.

The page offset field of the virtual address defines the byte within the page frame in which the virtual address is mapped. The physical address consists of the page frame base address from the page table entry concatenated with the page offset field of the virtual address. If a protection violation is detected, no memory access is made and a memory fault exception is executed.

Memory Management (continued)

Memory Management Operations

Both TLBs and NPSRs are completely flushed whenever the ATT92010 *Hobbit* microprocessor is reset (either by asserting the external reset pin, or the detection of an internal event which causes the *Hobbit* microprocessor to reset). The TLBs and NPSRs are also flushed whenever the segment table base register is written (see the STB—Segment Table Base section on page 39).

Individual TLB and NPSR entries may be flushed with the FLUSHPTE instruction. If the effective address in the FLUSHPTE instruction is cached in one or both of the TLBs or NPSRs, the TLB or NPSR entry is marked invalid, so any subsequent access of that virtual address will be translated by the full memory map table walk. The FLUSHPTE instruction is not privileged, so a user process may flush any or all entries in the on-chip TLBs or NPSRs. Although this may degrade the performance of the process, it does not affect correctness, since the memory management tables in physical memory define the address mapping and the FLUSHPTE instruction does not alter the tables in memory.

$\overline{\text{LOCK}}$ is asserted when page table entries are fetched. If the R and M bits of the entry are current, $\overline{\text{LOCK}}$ is cleared. If either R or M bits must be updated, the page table entry is written back to memory with $\overline{\text{LOCK}}$ still asserted. $\overline{\text{LOCK}}$ is deasserted when the write completes.

If there is an external bus error signaled during the memory management table walk, the *Hobbit* microprocessor will take an exception (see Table 22 on page 29).

Bus Operation and Arbitration

To facilitate multiple bus masters, the bus arbitration protocol does not make the ATT92010 the default bus master. A centralized arbiter selects the current bus master and controls transactions over the bus. A synchronous bus protocol is used to exchange ownership of the bus from one master to another. The central bus arbiter must execute this protocol, asserting and negating $\overline{\text{BGRANT}}$ to the various bus masters in a consistent manner.

The signals involved in this protocol generated by the central bus arbiter are $\overline{\text{HRESET}}$, $\overline{\text{BGRANT}}$, and $\overline{\text{RETRY}}$. There is a $\overline{\text{BGRANT}}$ for each bus master, with the other signals shared among bus masters.

The signals involved in this protocol generated by the bus masters are $\overline{\text{BREQ}}$, $\overline{\text{START}}$, IOC[1:0], and $\overline{\text{LOCK}}$. There is a $\overline{\text{BREQ}}$ for each bus master, with the other signals shared among bus masters.

Finally, the device being accessed generates $\overline{\text{DTACK}}$.

Upon reset of the system, which must be synchronous, the arbiter selects one of the requesting bus masters as current bus master by asserting its $\overline{\text{BGRANT}}$. Having received $\overline{\text{BGRANT}}$, the master takes ownership of the bus. The bus arbiter monitors the bus, keeping track of the state of the bus. The *Hobbit* microprocessor asserts $\overline{\text{BREQ}}$ when an I/O transaction is pending (upon reset, all *Hobbit* microprocessors want to start execution at address 0x0).

The arbiter selects a new bus master by deasserting $\overline{\text{BGRANT}}$ to the current bus master and asserting $\overline{\text{BGRANT}}$ to the next bus master at the end of any outstanding bus transactions. If the current bus master loses $\overline{\text{BGRANT}}$ with an outstanding transaction on the bus, that master remains on the bus until $\overline{\text{DTACK}}$ is asserted with IOC[1:0] equal to zero and $\overline{\text{LOCK}}$ is deasserted.

The new bus master takes ownership of the bus at the beginning of the next bus cycle after receipt of $\overline{\text{BGRANT}}$. The arbiter must assert $\overline{\text{BGRANT}}$ in a manner which inserts a dead cycle between the end of the previous bus owner's $\overline{\text{BGRANT}}$ and the beginning of the next bus owner's $\overline{\text{BGRANT}}$.

The ATTT92010 asserts $\overline{\text{BGACK}}$ to indicate that it has bus ownership, and it deasserts $\overline{\text{BGACK}}$ to indicate that it has relinquished the bus.

Requesting the Bus

In Figure 28, bus cycles 1 through 4 show a typical bus request and acquisition.

Surrendering the Bus

The arbiter signals the *Hobbit* microprocessor to relinquish the bus by deasserting $\overline{\text{BGRANT}}$. When $\overline{\text{BGRANT}}$ is deasserted, the ATT92010 will relinquish ownership of the bus and deassert $\overline{\text{BGACK}}$. If the *Hobbit* microprocessor is running a bus transaction and $\overline{\text{BGRANT}}$ is deasserted, ownership of the bus will be relinquished after receipt of $\overline{\text{DTACK}}$ with IOC[1:0] equal to zero and $\overline{\text{LOCK}}$ is deasserted. If the *Hobbit* microprocessor is not running a bus transaction and $\overline{\text{BGRANT}}$ is deasserted, ownership of the bus will be relinquished at the beginning of the next bus cycle. $\overline{\text{BGACK}}$ is deasserted by the *Hobbit* microprocessor in the same bus cycle that ownership of the bus is being relinquished.

Most arbitration protocols will want to continue to grant the bus to the current bus master if it continues to request the bus by asserting its $\overline{\text{BREQ}}$.

In Figure 28 on page 50, bus cycles 15 through 17 show a typical release of the bus.

Bus Operation and Arbitration (continued)

Surrendering the Bus (continued)

Figure 28 represents a cacheable single-word data read followed by a double-word text read. The accesses are not interlocked and don't produce bus errors.

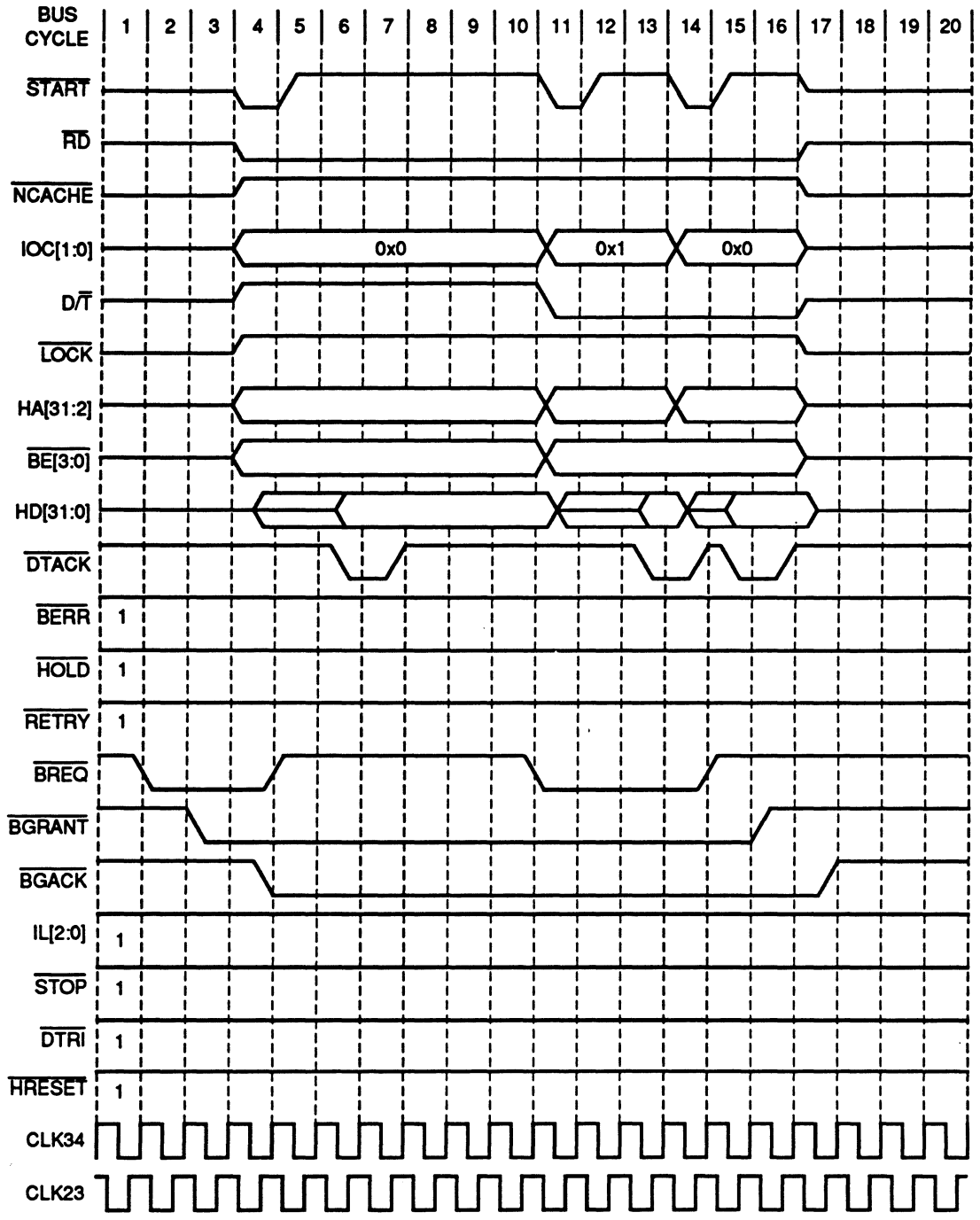


Figure 28. Read Bus Cycles with Bus Arbitration

Bus Operation and Arbitration

(continued)

Bus Transaction Types

Normal bus transfers begin with the assertion of $\overline{\text{START}}$ and end with the assertion of $\overline{\text{DTACK}}$. In case of an error during a bus transfer, the transaction may be ended by the assertion of $\overline{\text{HRESET}}$ or $\overline{\text{BERR}}$ with $\overline{\text{DTACK}}$. Interlocked bus transfers end with the deassertion of $\overline{\text{LOCK}}$ following a $\overline{\text{DTACK}}$. Multiple word transfers end when IOC[1:0] = 0 with assertion of $\overline{\text{DTACK}}$. Sub-word accesses are the same as single-word accesses with the exception that only the appropriate byte enables are asserted.

Read Transactions

Read transactions may fetch text or data. Text reads are always double-word transfers. Data reads are either single-, double- or quad-word transfers. After completion of a read transaction, a loopback is performed if the *Hobbit* microprocessor remains owner of the bus and there are no pending bus transactions. See Figure 28 on page 50 for the following example.

Bus cycles 4 through 6 show a typical read transaction. In bus cycles 7 through 10, a loopback cycle is performed. In bus cycles 11 and 15, a double-word transaction is performed. In bus cycle 16, another loopback cycle is performed. The ATT92010 holds all bus signals at their previous values and loops back the data read on the previous cycle.

Note: The bus transaction may be ended by $\overline{\text{HRESET}}$ or $\overline{\text{BERR}}$ with $\overline{\text{DTACK}}$ to signal an error.

Write Transactions

Write transactions are either single-, double-, or quad-word transfers. See Figure 29 on page 52 for the following example.

Bus cycles 4 through 7 show a typical write transaction. In bus cycles 8 through 10, the microprocessor maintains the previous bus cycles values on most signals. In bus cycles 11 through 13 and bus cycles 14 through 15, two more write transactions are performed.

Note: The bus transaction may be ended by $\overline{\text{HRESET}}$ or $\overline{\text{BERR}}$ with $\overline{\text{DTACK}}$ to signal an error.

Interlocked Bus Transfer

This is a read-modify-write type bus operation. This sequence of operations is noninterruptible. The bus remains locked through the write. If $\overline{\text{BGRANT}}$ is de-asserted during an interlocked operation, the operation is completed and transfer of bus ownership is delayed a clock cycle. See Figure 30 on page 53 for the following example.

Bus cycles 2 through 4 show the read portion of the RMW operation. Bus cycles 7 through 8 show the write portion of the RMW operation. In bus cycle 9, $\overline{\text{LOCK}}$ remains asserted by the microprocessor adding a dead cycle. In bus cycle 11, the next bus cycle begins.

Block Data Transfer

The block transfer sizes that are supported are double- and quad-word. The block transfer looks like a series of single-word bus transfers with the microprocessor incrementing address bits HA[3:2] and decrementing IOC[1:0] for each access. Block transfers are not interruptible. Block transfers may be retried with the transfer resuming where it was aborted when $\overline{\text{RETRY}}$ is deasserted.

Bus Operation and Arbitration (continued)

Bus Transaction Types (continued)

Figure 29 represents a cacheable single-word data write followed by a double-word data write.

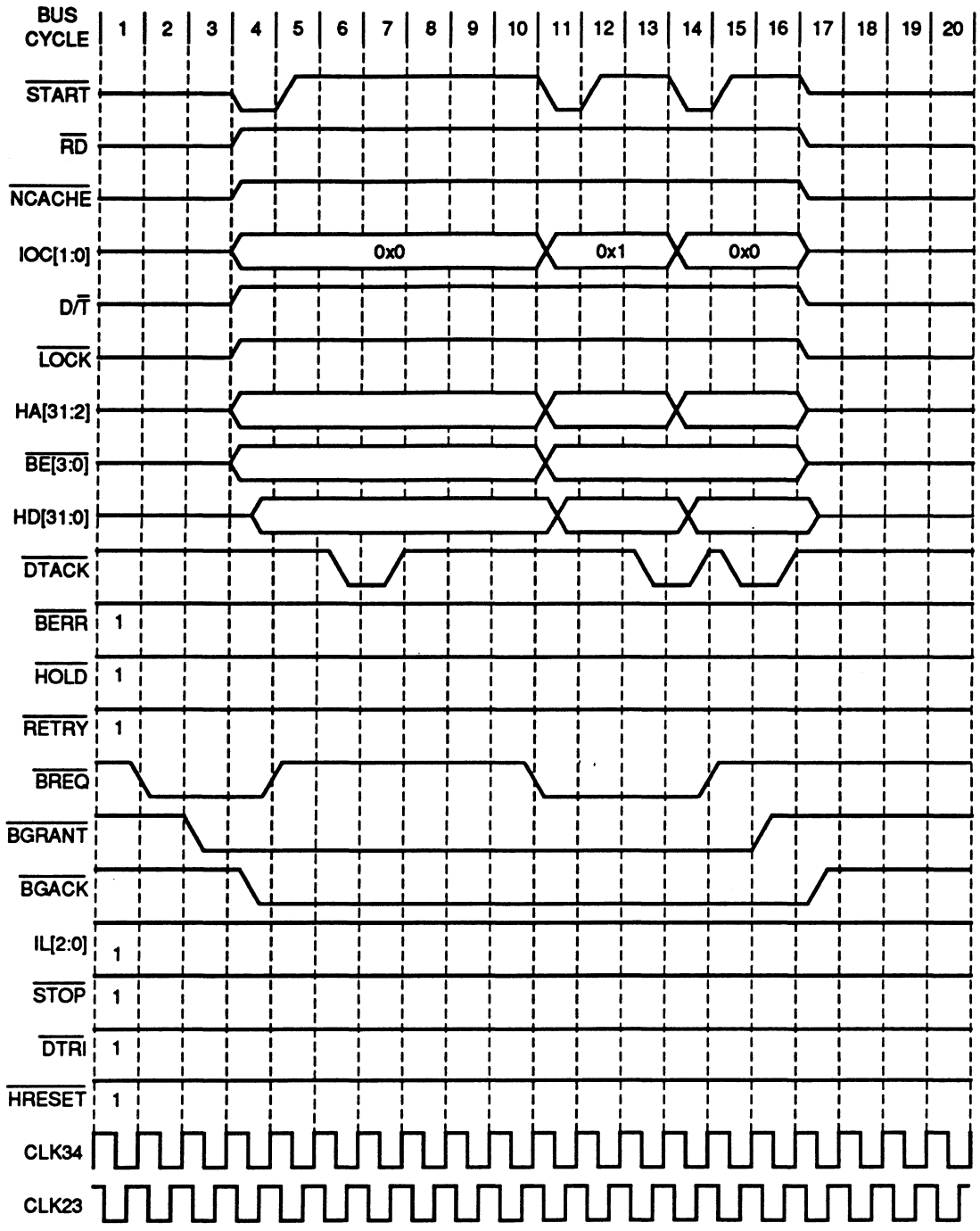


Figure 29. Write Bus Cycles with Bus Arbitration

Bus Operation and Arbitration (continued)

Bus Transaction Types (continued)

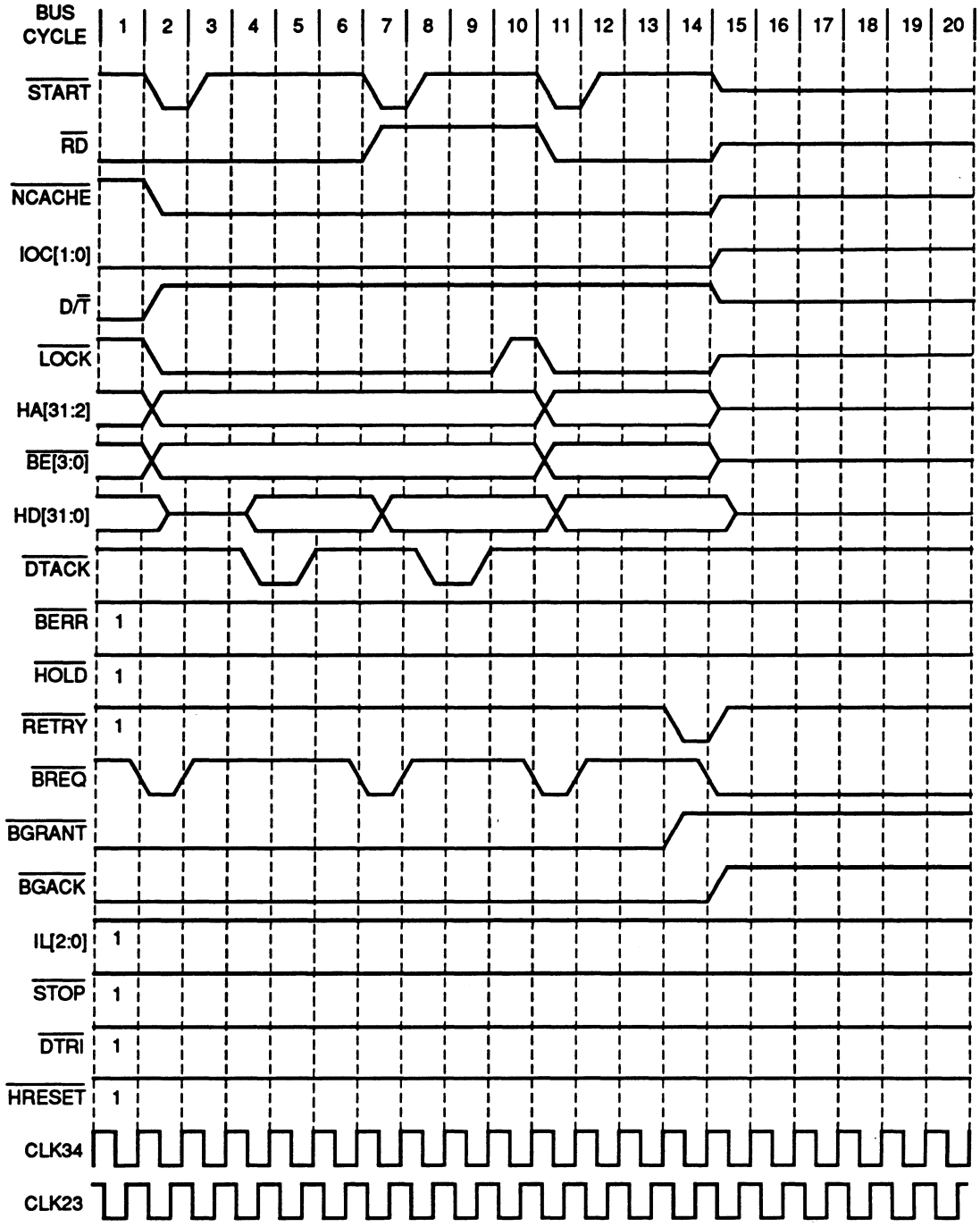


Figure 30. Interlocked Bus Transfer without and with Retry

Bus Operation and Arbitration

(continued)

Exception Handling

The exception/error signals provide a means by which external devices can inform the ATT92010 of an unusual condition which requires the processor to deviate from its normal execution.

Bus Retry

$\overline{\text{RETRY}}$ is asserted to retry the current bus transaction. When $\overline{\text{RETRY}}$ is asserted during a valid bus transaction, the *Hobbit* microprocessor aborts the current bus transfer and masks the $\overline{\text{DTACK}}$ input. After $\overline{\text{RETRY}}$ is deasserted, the bus transaction is rerun after the *Hobbit* microprocessor obtains ownership of the bus as $\overline{\text{RETRY}}$ is orthogonal to bus arbitration. In systems with gateways through which two buses communicate with each other, the retry feature is required to break deadlock conditions when the two buses have simultaneous requests for their respective counterpart bus.

Bus Error

The assertion of $\overline{\text{BERR}}$ indicates an error in a bus transaction of any type. An internal I/O fault is generated when $\overline{\text{BERR}}$ is asserted and a $\overline{\text{DTACK}}$ is received. When $\overline{\text{BERR}}$ is asserted and $\overline{\text{DTACK}}$ received, the exception taken depends upon the type of bus transaction being terminated.

ATT92010 Reset

If $\overline{\text{HRESET}}$ is asserted, the ATT92010 is reset and any current bus cycle is aborted.

Table 40. Bus Transaction Termination Signals Priority Levels

Signal	Priority Level
$\overline{\text{HRESET}}$	Highest
$\overline{\text{RETRY}}$	↓
$\overline{\text{DTACK}}$	Lowest

Testability

The *Hobbit* microprocessor is a highly testable design providing access to all testability features via the IEEE 1149.1/D5 interface. The features that are accessible include:

- Single clock delay by-pass.
- Boundary-scan of I/O signals.
- Embedded memory built-in test (BIT) and scan features.
- Embedded PLA BIT features.

Conformance

The test access port (TAP) provided conforms to all aspects of the IEEE 1149.1/D5 except for TCK and $\overline{\text{TRST}}$.

In IEEE 1149.1/D5, TCK is required to be a free-running clock with any gating performed within the device. This feature is not provided; therefore, TCK must be gated externally. Also, an unconnected $\overline{\text{TRST}}$ is to be terminated in the inactive mode. In the ATT92010, an unconnected $\overline{\text{TRST}}$ is internally terminated in the active mode holding the TAP state-machine in reset.

TAP Controller (TAPC)

The TAPC is a synchronous finite state machine whereby sequencing through the various operations of the testability circuitry occurs under control of the TMS signal.

The state diagram for the TAPC is shown in Figure 31 on page 56. There are 16 states in this state machine with advancement of state dependent upon the value of TMS at the rising edge of TCK. All operations of the test logic occur on the rising edge of TCK following the entry into a controller state. Changes at TDO occur on the falling edge of TCK following entry into a controller state which selects TDO. The states of the TAPC are defined in Table 41.

Testability (continued)

TAP Controller (TAPC) (continued)

Table 41. TAP Controller State Table

State	Description
0x0	**Exit(2)-DR.** This is a temporary controller state. All test data registers and the instruction register retain their previous state. A high signal on the TMS line while in this state causes termination of the scanning process; a low causes entry into the Shift-DR state.
0x1	**Exit(1)-DR.** This is a temporary controller state. All test data registers and the instruction register retain their previous state. TMS = 1 in this state causes termination of the scanning process; TMS = 0 causes entry into the Pause-DR state.
0x2	**Shift-DR.** In this controller state, the selected data register shifts data one stage towards its serial output on each rising edge of TCK. All registers other than the selected test data register retain their previous state.
0x3	**Pause-DR.** This controller state allows shifting of the selected test data register to be temporarily halted. All test data registers and the instruction register retain their previous state. The controller remains in this state while TMS = 0. When TMS goes high, the controller advances to the Exit(2)-DR state.
0x4	**Select-IR-Scan.** This is a temporary controller state in which all test logic retains its previous state. If TMS = 0 when the controller is in this state, then a scan sequence for the instruction register is initiated.
0x5	**Update-DR.** During this controller state, data is transferred from each shift-register stage into the corresponding parallel output latch (if the selected test data register includes a parallel output latch). All shift-register stages in the selected register retain their previous state.
0x6	**Capture-DR.** In this controller state, data is parallel loaded into the selected test data register. If the register does not have a parallel input, or if capturing is not required for the selected test, the register retains its previous state unchanged.
0x7	**Select-DR-Scan.** This is a temporary controller state in which all test logic retains its previous state. If TMS = 0 when the controller is in this state, then a scan sequence for the selected test data register is initiated.
0x8	**Exit(2)-IR.** This is a temporary controller state. All test data registers and the instruction register retain their previous state. A high signal on the TMS line while in this state causes termination of the scanning process; a low causes entry into the Shift-IR state.
0x9	**Exit(1)-IR.** This is a temporary controller state. All test data registers and the instruction register retain their previous state. If TMS = 1 while in this state, the scanning process is terminated; if 0, the Pause-IR state is entered.
0xA	**Shift-IR.** In this controller state, the instruction register shifts data one stage towards its serial output on each rising edge of TCK.
0xB	**Pause-IR.** This controller state allows shifting of the instruction register to be temporarily halted. All test data registers and the instruction register retain their previous state. The controller remains in this state while TMS = 0. When TMS goes high, the controller advances to the Exit(2)-DR state.
0xC	**Run-Test/Idle.** The controller state between scan operations where an internal test previously selected by setting the instruction register may be executed. Registers not involved in the application of the test retain their previous state. If the data in the instruction register does not indicate that a test should be executed, then all test logic must retain their previous state. Once entered, the controller will remain in the Run-Test/Idle state as long as TMS = 0.
0xD	**Update-IR.** During this controller state, the instruction is transferred from each shift-register stage of the instruction register into the parallel output latch of the instruction register. All shift-register stages in the instruction register retain their previous state.
0xE	**Capture-IR.** In this controller state, data is parallel loaded into the instruction register. If the register does not have a parallel input, or if capturing is not required for the selected test, the register retains its previous state unchanged.
0xF	**Test-Logic-Reset.** While in this state, all test circuitry is disabled. The instruction register (IR) is reset to select the by-pass register. The controller remains in this state as long as TMS = 1 or $\overline{\text{TRST}}$ is asserted.

Testability (continued)

TAP Controller (TAPC) (continued)

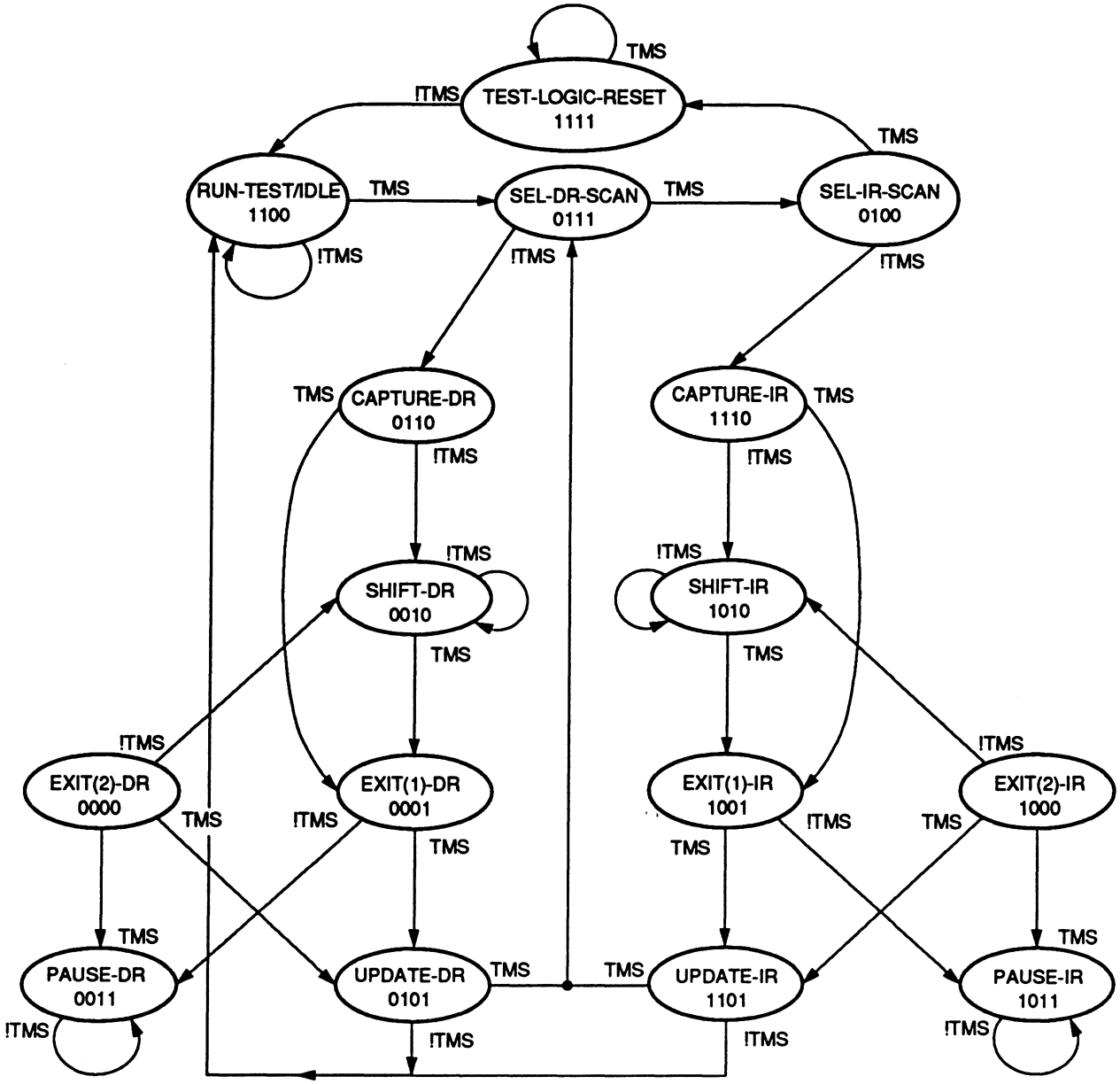


Figure 31. TAP Controller State Diagram

Testability (continued)

IEEE Registers 1149.1/D5 Registers

The following registers are described in the IEEE 1149.1/D5 specification.

TAP Instructions and Instruction Register

The instruction register (IR) allows a test instruction to be shifted into the ATT92010 *Hobbit* microprocessor. The IR is used to select the test to be performed or the test data register to be accessed. The IR is 7 bits in length. Table 42 identifies the instruction encodings.

Table 42. The ATT92010 JTAG/1149.1 TAP Instruction Register Encodings

Instruction	Register Selected	Instruction Mnemonic	Description
0000000	BS	EXTEST	BS selected with BS external test.
0000001	BS	SAMPLE	BS selected with BS sample.
0000010	BS	INTEST	BS selected with BS internal test.
0000011	PPLA	IRPPLA	PPLA selected with PPLA self-test.
0000100	ICD	IRICD	Instruction cache data selected with ICD self-test.
0000101	SC	IRSC	Stack cache selected with SC self-test.
0000110	PFD	IRPFD	Prefetch cache data selected with PFD self-test.
0000111	PFT	IRPFT	Prefetch cache tag selected with PFT self-test.
0001xxx	NA	NA	Reserved.
001xxxx	BP	BP	BP selected with all self-test.
01xxxxx	BP	BP	BP selected and BS sample.
10xxxxx	ID	ID	ID selected and BS sample.
11xxxxx	BP	BP	BP selected and BS sample.

Boundary-Scan Register

The boundary-scan register (BS) allows testing of circuitry external to the *Hobbit* microprocessor. Additionally, BS provides for sampling and examination of the I/O values without impacting the operation of the system logic. Ninety shift elements are in the boundary-scan shift chain. Ninety-one TCKs are required to shift the entire chain from TDI through to TDO. Position is given from TDI to TDO.

Table 43. Boundary-Scan Shift Chain

Position	Name	Description	Position	Name	Description
1	$\overline{\text{HRESET}}$	I	2	CLK23	Sample Only I
3	$\overline{\text{STOP}}$	Sample Only I	4	CLK34	Sample Only I
5	$\overline{\text{DTRI}}$	I	6	3-data	Control for I/O
7	D31	I/O	8	HD30	I/O
9	HA22	3-State O	10	HD4	I/O
11	HA15	3-State O	12	HD3	I/O
13	HA3	3-State O	14	HD29	I/O
15	HA14	3-State O	16	HD28	I/O
17	HA2	3-State O	18	HD27	I/O

Testability (continued)

IEEE Registers 1149.1/D5 Registers (continued)

Table 43. Boundary-Scan Shift Chain (continued)

Position	Name	Description	Position	Name	Description
19	HA13	3-State O	20	HD26	I/O
21	HA31	3-State O	22	HD25	I/O
23	HA30	3-State O	24	HD24	I/O
25	HA29	3-State O	26	HD23	I/O
27	HA12	3-State O	28	HD22	I/O
29	HA21	3-State O	30	HD21	I/O
31	HA11	3-State O	32	HD20	I/O
33	HA20	3-State O	34	HD7	I/O
35	HA10	3-State O	36	HD6	I/O
37	HA19	3-State O	38	HD5	I/O
39	HA9	3-State O	40	HD19	I/O
41	HA28	3-State O	42	HD18	I/O
43	HA8	3-State O	44	HD17	I/O
45	HA27	3-State O	46	HD16	I/O
47	HA26	3-State O	48	HD15	I/O
49	HA25	3-State O	50	HD14	I/O
51	HA7	3-State O	52	HD13	I/O
53	HA18	3-State O	54	HD2	I/O
55	HA6	3-State O	56	HD1	I/O
57	HA17	3-State O	58	HD0	I/O
59	HA5	3-State O	60	HD12	I/O
61	HA16	3-State O	62	HD11	I/O
63	HA4	3-State O	64	HD10	I/O
65	HA24	3-State O	66	HD9	I/O
67	HA23	3-State O	68	HD8	I/O
69	D/$\overline{T}$	3-State O	70	$\overline{NCACHE}$	3-State O
71	$\overline{RD}$	3-State O	72	$\overline{BE0}$	3-State O
73	$\overline{BE1}$	3-State O	74	$\overline{BE2}$	3-State O
75	$\overline{BE3}$	3-State O	76	IOC1	3-State O
77	IOC0	3-State O	78	$\overline{LOCK}$	3-State O
79	START	3-State O	80	$\overline{BGACK}$	O
81	$\overline{BREQ}$	O	82	3-bus	Control for 3-State O
83	$\overline{BGRANT}$	I	84	$\overline{RETRY}$	I
85	$\overline{BERR}$	I	86	$\overline{HOLD}$	I
87	$\overline{DTACK}$	I	88	IL2	I
89	IL1	I	90	IL0	I

Notes:
3-data and 3-bus control the 3-stating of the output side of the data pins and output pins, respectively. A 1 3-states, and a 0 enables.

Position 1, the $\overline{HRESET}$ bit, is closest to TDI. Position 90, the IL0 bit, is closest to TDO.

Testability (continued)

IEEE Registers 1149.1/D5 Registers (continued)

Identification Register

See page 34 for a description of the identification register (ID). The ID register is accessible through both the TAP and normal register access.

By-Pass Register

The by-pass (BP) register provides a single TCK delay path from TDI to TDO. When the BP register is selected, a 0 is loaded on the rising edge of TCK in the Capture-DR controller state. When the Test-Logic-Reset controller state is entered, the BP register retains its last value.

Absolute Maximum Ratings

Stresses in excess of the Absolute Maximum Ratings can cause permanent damage to the device. These are absolute stress ratings only. Functional operation of the device is not implied at these or any other conditions in excess of those given in the operational sections of the data sheet. Exposure to Absolute Maximum Ratings for extended periods can adversely affect device reliability.

Parameter	Symbol	Min	Max	Unit
Storage Temperature	T_{stg}	–40	125	°C
Supply Voltage	V_{DD}	–0.5	7.0	V
Input Voltage	V_{IN}	V_{SS} – 0.5	V_{DD} + 0.5	V
Ambient Operating Temperature	T_A	0	70	°C

Handling Precautions

All MOS devices must be handled with certain precautions to avoid damage due to the accumulation of static charge. Although input protection circuitry has been incorporated into the devices to minimize the effect of this statics buildup, proper precautions should be taken to avoid exposure to electrostatic discharge (ESD) during handling and mounting. AT&T employs a human-body model (HBM) for ESD susceptibility testing. Since the failure voltage of electrostatic devices is dependent on the current and voltage and, hence, the resistance and capacitance, it is important that standard values be employed to establish a reference by which to compare test data. Values of 100 pF and 1500 Ω are the most common and are the values used in the AT&T HBM test circuit. The breakdown voltage for the *Hobbit* microprocessor is 1,000 V, according to the HBM, and it is 2,000 V, according to the charged-device model (CDM).

Electrical Characteristics

The parameters in Tables and are valid for: T_A = 0 °C to 70 °C

Table 44. Recommended Operating Conditions (V_{DD} = 3.3 V ± 10%; CLK34 and CLK23 = 20 MHz)

Parameter	Symbol	Min	Typ	Max	Unit
Input High Voltage	V_{IH}	2.2	—	V_{DD} + 0.3	V
Input Low Voltage	V_{IL}	–0.3	—	0.6	V
Output High Voltage I_{OH} = 5 mA (pins HD[31:0]) I_{OH} = 2 mA (all pins except HD[31:0])	V_{OH}	2.5	—	—	V
Output Low Voltage I_{OL} = 5 mA (pins HD[31:0]) I_{OL} = 2 mA (all pins except HD[31:0])	V_{OL}	—	—	0.3	V
TDI Input Low Current	I_{TDI}	—	—	–1.73	mA
TMS Input Low Current	I_{TMS}	—	—	–0.87	mA
TCK Input Low Current	I_{TCK}	—	—	–0.87	mA
$\overline{TRST}$ Input High Current	I_{TRST}	—	—	–1.16	mA
Input Leakage Current 0 V ≤ V_{IN} ≤ V_{DD}	I_I	–0.01	—	0.01	mA
3-stated Output Leakage Current	I_{OTI}	–0.01	—	0.01	mA
Supply Current Output Load = 10 pF Output Load = 50 pF	 I_{DD} I_{DD}	 — —	 75 125	 95 150	 mA mA
Standby Current	I_{SB}	0	0.006	0.030	mA

Table 45. Recommended Operating Conditions (V_{DD} = 5.0 V ± 10%; CLK34 and CLK23 = 30 MHz)

Parameter	Symbol	Min	Typ	Max	Unit
Input High Voltage	V_{IH}	3.2	—	V_{DD} + 0.4	V
Input Low Voltage	V_{IL}	–0.4	—	0.8	V
Output High Voltage I_{OH} = 7 mA (pins HD[31:0]) I_{OH} = 3 mA (all pins except HD[31:0])	V_{OH}	4.0	—	—	V
Output Low Voltage I_{OL} = 7 mA (pins HD[31:0]) I_{OL} = 3 mA (all pins except HD[31:0])	V_{OL}	—	—	0.4	V
TDI Input Low Current	I_{TDI}	—	—	–2.63	V
TMS Input Low Current	I_{TMS}	—	—	–1.31	mA
TCK Input Low Current	I_{TCK}	—	—	–1.31	mA
$\overline{TRST}$ Input High Current	I_{TRST}	—	—	–1.75	mA
Input Leakage Current 0 V ≤ V_{IN} ≤ V_{DD}	I_I	–0.01	—	0.01	mA
3-stated Output Leakage Current	I_{OTI}	–0.01	—	0.01	mA
Supply Current Output Load = 10 pF Output Load = 50 pF	 I_{DD} I_{DD}	 — —	 175 285	 220 340	 mA mA
Standby Current	I_{SB}	—	0.009	0.050	mA

Timing Characteristics

All timing is based on a 50 pF load under worst-case conditions, although the device is capable of driving heavier loads.

Load Specifications

Table 46. Capable Loading Specifications, Test Loading, and Output Derating Factors

Output Signal	Max Load (pF)	Test Load (pF)	Output Derating (ns/pF)	
			3.3 V	5.0 V
HA[31:2]	100	50	0.09	0.06
$\overline{\text{BGACK}}$	100	50	0.09	0.06
$\overline{\text{BE[3:0]}}$	100	50	0.09	0.06
$\overline{\text{BREQ}}$	100	50	0.09	0.06
D/$\overline{\text{T}}$	100	50	0.09	0.06
IOC[1:0]	100	50	0.09	0.06
$\overline{\text{LOCK}}$	100	50	0.09	0.06
$\overline{\text{NCACHE}}$	100	50	0.09	0.06
$\overline{\text{START}}$	100	50	0.09	0.06
TDO	100	50	0.09	0.06
$\overline{\text{RD}}$	100	50	0.09	0.06
HD[31:0]	150	50	0.06	0.04

The output derating factors may be used to obtain an approximate rate of increase of output valid delay time with increasing load capacitance up to the maximum loading specified.

Timing Characteristics (continued)

Timing Diagrams

The following figures give timing specifications.

Clock

Two 1x clocks in quadrature are required by the ATT92010 *Hobbit* microprocessor. The internal clocks are decoded from these inputs. The internal clocks can be stopped in phase 1 by asserting $\overline{\text{STOP}}$ prior to phase 1 allowing for burst-mode, single-stepping, and suspended operation.

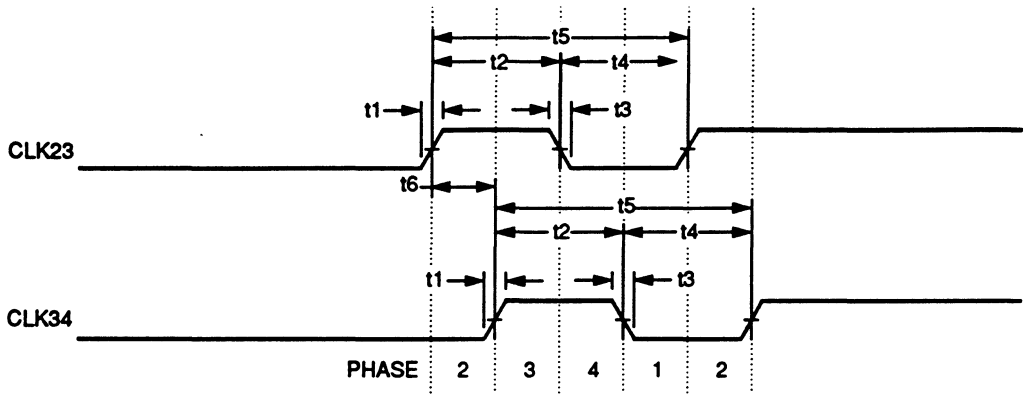


Figure 32. Clock Input Timing

Table 47. Clock Input Timing

Symbol	Parameter	3.3 V		5.0 V		Unit
		Min	Max	Min	Max	
t1	Rise Time	—	3.0	—	3.0	ns
t2	Pulse High	22.5	27.5	14.5	18.5	ns
t3	Fall Time	—	3.0	—	3.0	ns
t4	Pulse Low	22.5	27.5	14.5	18.5	ns
t5	Period	50.0	100	33.3	50.0	ns
t6	Delay	10.5	14.5	6.3	10.2	ns

Timing Characteristics (continued)

Timing Diagrams (continued)

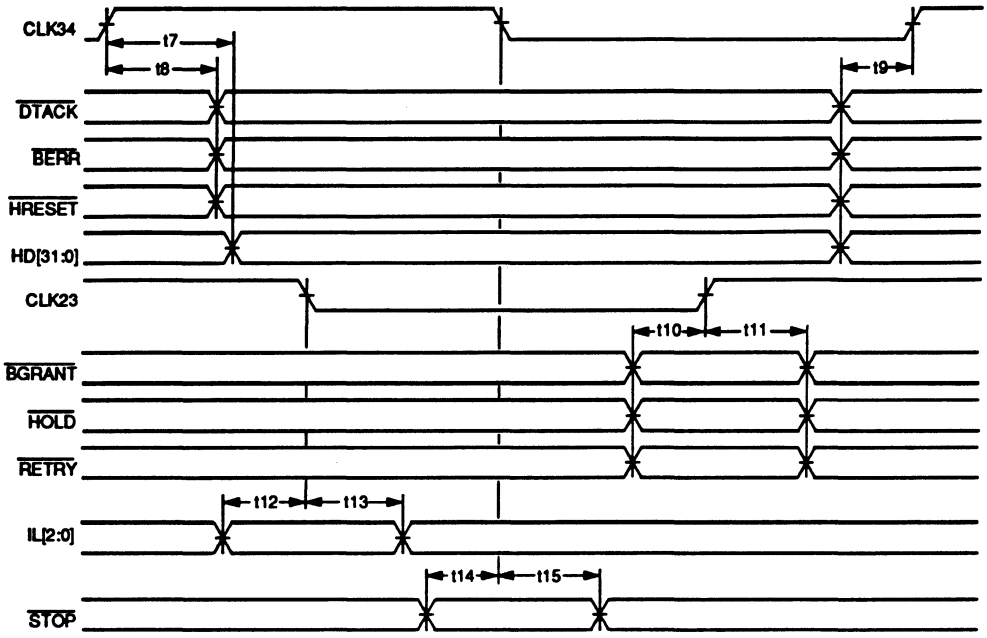


Figure 33. Synchronous Input Timing

Table 48. Synchronous Input Timing

Symbol	Signal	Type	Reference	3.3 V		5.0 V		Unit
				Min	Max	Min	Max	
t7	HD[31:0]	Input Hold	CLK34 Rise	7.0	—	6.0	—	ns
t8	$\overline{\text{DTACK}}$	Input Hold	CLK34 Rise	5.0	—	4.0	—	ns
	$\overline{\text{BERR}}$	Input Hold	CLK34 Rise	5.0	—	4.0	—	ns
	$\overline{\text{HRESET}}$	Input Hold	CLK34 Rise	5.0	—	4.0	—	ns
t9	HD[31:0]	Input Setup	CLK34 Rise	3.0	—	2.0	—	ns
	$\overline{\text{DTACK}}$	Input Setup	CLK34 Rise	3.0	—	2.0	—	ns
	$\overline{\text{BERR}}$	Input Setup	CLK34 Rise	3.0	—	2.0	—	ns
	$\overline{\text{HRESET}}$	Input Setup	CLK34 Rise	3.0	—	2.0	—	ns
t10	$\overline{\text{BGRANT}}$	Input Setup	CLK23 Rise	3.0	—	2.0	—	ns
	$\overline{\text{HOLD}}$	Input Setup	CLK23 Rise	3.0	—	2.0	—	ns
	$\overline{\text{RETRY}}$	Input Setup	CLK23 Rise	3.0	—	2.0	—	ns
t11	$\overline{\text{BGRANT}}$	Input Hold	CLK23 Rise	5.0	—	4.0	—	ns
	$\overline{\text{HOLD}}$	Input Hold	CLK23 Rise	5.0	—	4.0	—	ns
	$\overline{\text{RETRY}}$	Input Hold	CLK23 Rise	5.0	—	4.0	—	ns
t12	IL[2:0]	Input Setup	CLK23 Fall	3.0	—	2.0	—	ns
t13	IL[2:0]	Input Hold	CLK23 Fall	5.0	—	4.0	—	ns
t14	$\overline{\text{STOP}}$	Input Setup	CLK34 Fall	5.0	—	4.0	—	ns
t15	$\overline{\text{STOP}}$	Input Hold	CLK34 Fall	3.0	—	2.0	—	ns

Timing Characteristics (continued)

Timing Diagrams (continued)

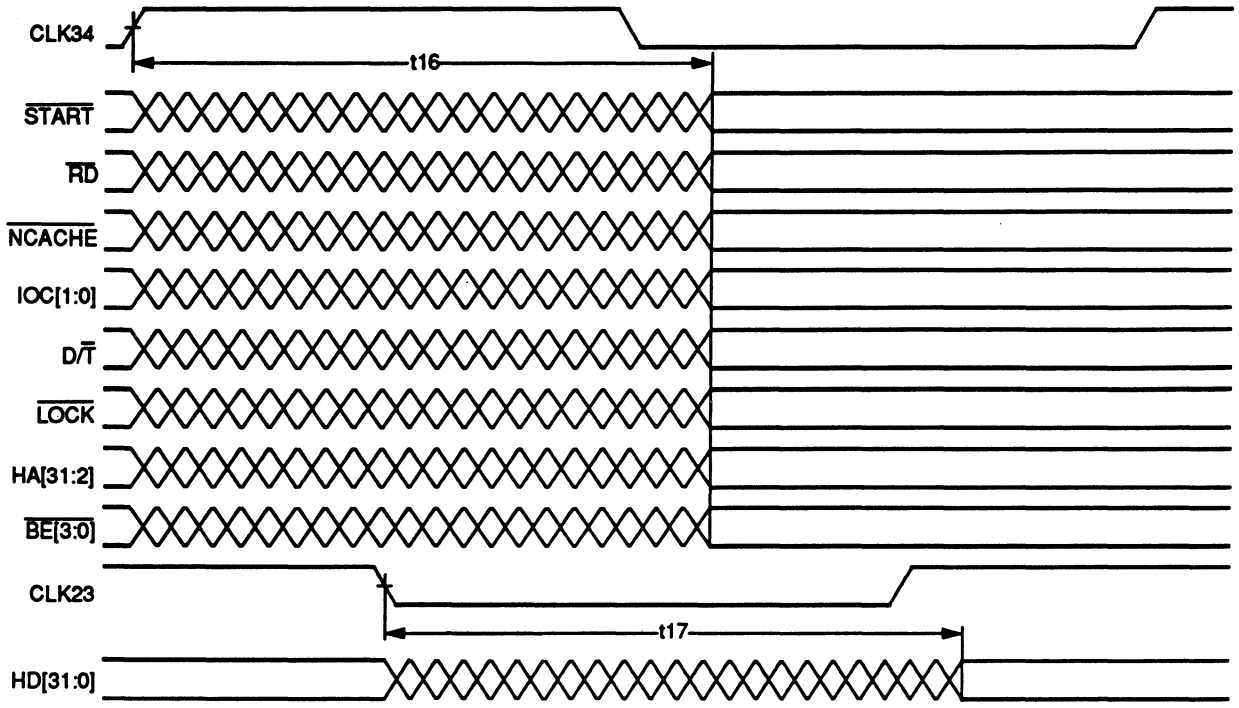


Figure 34. Output Timing

Table 49. Output Timing*

Symbol	Signal	Type	Reference	3.3 V		5.0 V		Unit
				Min	Max	Min	Max	
t16	START	Output Valid	CLK34 Rise	—	23	—	18	ns
	RD	Output Valid	CLK34 Rise	—	23	—	18	ns
	NCACHE	Output Valid	CLK34 Rise	—	23	—	18	ns
	IOC[1:0]	Output Valid	CLK34 Rise	—	23	—	18	ns
	D/T	Output Valid	CLK34 Rise	—	23	—	18	ns
	LOCK	Output Valid	CLK34 Rise	—	23	—	18	ns
	HA[31:2]	Output Valid	CLK34 Rise	—	19	—	15	ns
	BE[3:0]	Output Valid	CLK34 Rise	—	23	—	18	ns
t17	HD[31:0]	Output Valid	CLK23 Fall	—	24	—	19	ns

*Tested at 50 pF load.

Timing Characteristics (continued)

Timing Diagrams (continued)

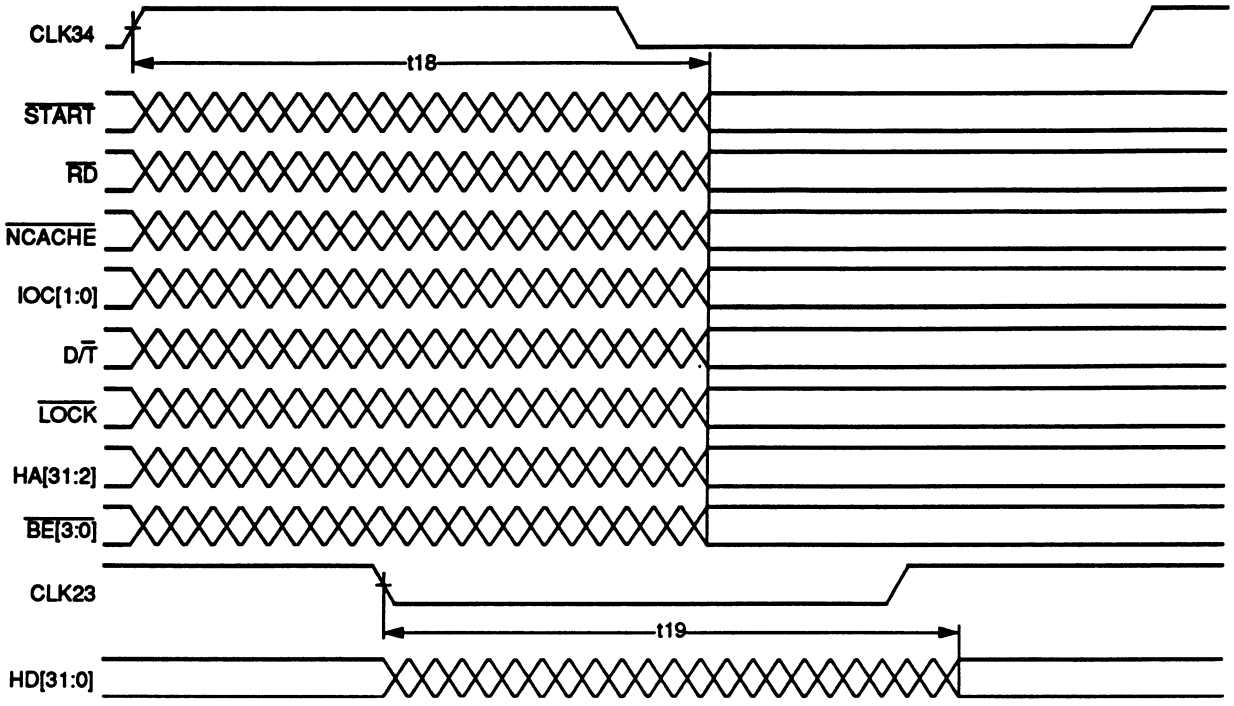


Figure 35. Bus Relinquish Cycle Output Timing

Table 50. Bus Relinquish Cycle Output Timing

Symbol	Signal	Type	Reference	3.3 V		5.0 V		Unit
				Min	Max	Min	Max	
t18	START	Output Hi-Z	CLK34 Rise	—	26	—	22	ns
	RD	Output Hi-Z	CLK34 Rise	—	26	—	22	ns
	NCACHE	Output Hi-Z	CLK34 Rise	—	26	—	22	ns
	IOC[1:0]	Output Hi-Z	CLK34 Rise	—	26	—	22	ns
	D/T	Output Hi-Z	CLK34 Rise	—	26	—	22	ns
	LOCK	Output Hi-Z	CLK34 Rise	—	26	—	22	ns
	HA[31:2]	Output Hi-Z	CLK34 Rise	—	26	—	22	ns
	BE[3:0]	Output Hi-Z	CLK34 Rise	—	26	—	22	ns
t19	HD[31:0]	Output Hi-Z	CLK23 Fall	—	26	—	22	ns

Timing Characteristics (continued)

Timing Diagrams (continued)

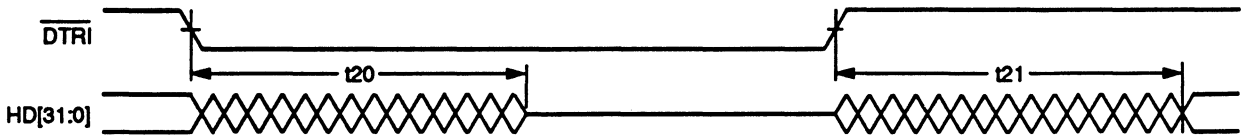


Figure 36. DTRI to Data Output Timing

Table 51. DTRI to Data Output Timing*

Symbol	Signal	Type	Reference	3.3 V		5.0 V		Unit
				Min	Max	Min	Max	
t20	HD[31:0]	Output Hi-Z	DTRI Fall	—	26	—	22	ns
t21	HD[31:0]	Output Valid	DTRI Rise	—	24	—	19	ns

*Tested at 50 pF load.

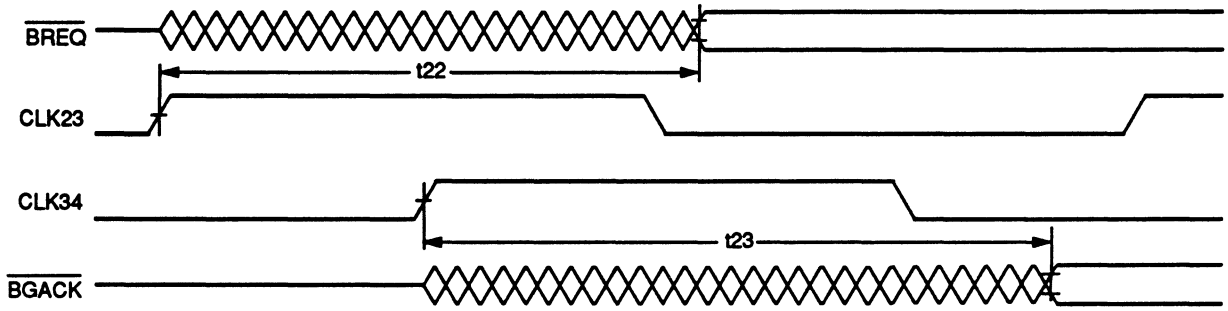


Figure 37. BREQ and BGACK Output Timing

Table 52. BREQ and BGACK Output Timing*

Symbol	Signal	Type	Reference	3.3 V		5.0 V		Unit
				Min	Max	Min	Max	
t22	BREQ	Output Valid	CLK23 Rise	—	23	—	18	ns
t23	BGACK	Output Valid	CLK34 Rise	—	32	—	25	ns

*Tested at 50 pF load.

Timing Characteristics (continued)

Timing Diagrams (continued)

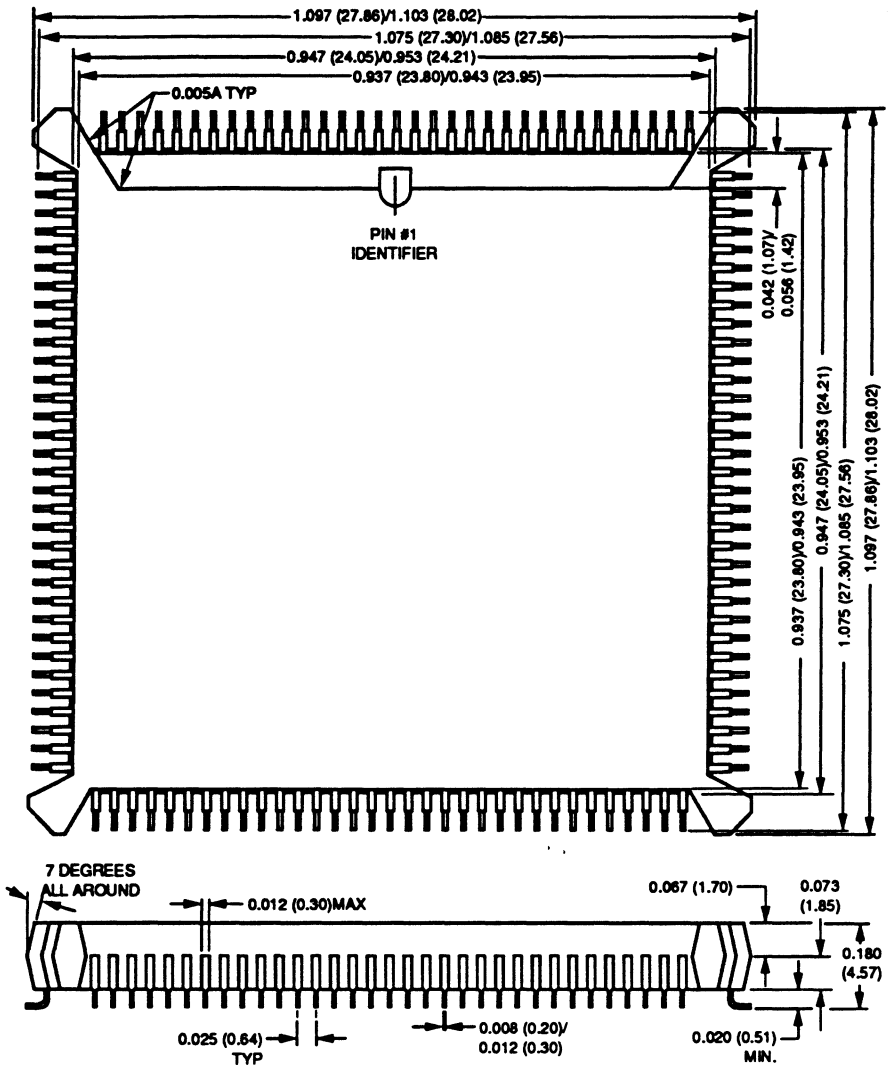
Table 53. JTAG Bus Timing Specifications

Signal	Type	Reference	3.3 V		5.0 V		Unit
			Min	Max	Min	Max	
TCK	Period	—	400.0	—	200.0	—	ns
TCK	Pulse High	—	200.0	—	100.0	—	ns
TCK	Pulse Low	—	200.0	—	100.0	—	ns
TDI	Input Setup	TCK rise	50.0	—	25.0	—	ns
TDI	Input Hold	TCK rise	50.0	—	25.0	—	ns
TMS	Input Setup	TCK rise	50.0	—	25.0	—	ns
TMS	Input Hold	TCK rise	50.0	—	25.0	—	ns
TDO	Output Valid	TCK fall	—	100.0	—	50.0	ns
TDO	Output Hi-Z	TCK fall	—	100.0	—	50.0	ns

Outline Diagram

132-pin PQFP

All dimensions are in inches and (millimeters).



For additional information, contact your AT&T Account Manager or the following:

U.S.A.: AT&T Microelectronics, Dept. AL-500404200, 555 Union Boulevard, Allentown, PA 18103
1-800-372-2447, FAX 215-778-4106 (In CANADA: **1-800-553-2448**, FAX 215-778-4106)

ASIA PACIFIC: AT&T Microelectronics Asia/Pacific, 14 Science Park Drive, #03-02A/04 The Maxwell, Singapore 0511
Tel. (65) 778-8833, FAX (65) 777-7495, Telex RS 42898 ATTM

JAPAN: AT&T Microelectronics, AT&T Japan Ltd., 7-18, Higashi-Gotanda 2-chome, Shinagawa-ku, Tokyo 141, Japan
Tel. (81) 3-5421-1600, FAX (81) 3-5421-1700

For data requests in Europe:

AT&T DATALINE: **Tel. (44) 732 742 999**, FAX (44) 732 741 221

For technical inquires in Europe:

CENTRAL EUROPE: **(49) 89 950 860** (Munich), NORTHERN EUROPE: **(44) 344 48711** (Bracknell UK), FRANCE: **(33) 47 67 47 67**,
SOUTHERN EUROPE: **(39) 266 011 800** (Milan) or **(34) 1 807 1441** (Madrid)

December 1992
DS91-214MCP

AT&T Microelectronics