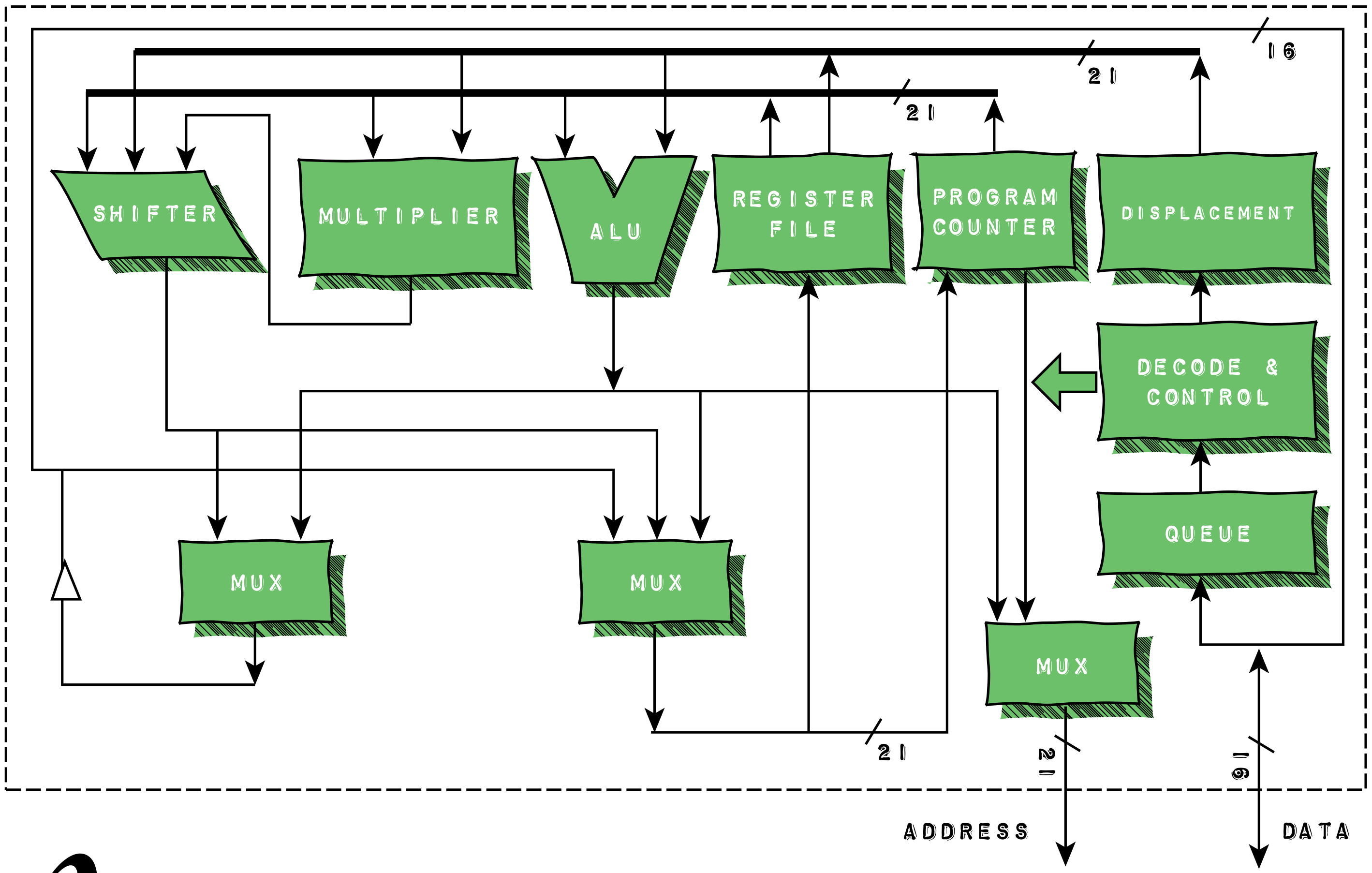
NATIONAL SEMICONDUCTOR CompactRISC™ 16-BIT ARCHITECTURE

INTRODUCTION

CompactRISC processor cores are designed specifically for embedded applications. The CompactRISC architecture is available in a wide range of implementations, supported by a common set of software development tools from multiple vendors. The CompactRISC architecture is scalable over a range of 8-, 16-, 32- and 64-bit processors, with common: variable-length instruction set, registers, addressing modes, interrupt and trap handling, debug support, and non-aligned data accesses, and efficient HLL execution.

FEATURES

- Available in Synethesizeable Verilog HDL
- Less than 1mm^2 @ 0.35µ
- 2Mbytes of linear address space
- Less than 0.3mA per MHZ @ 3 Volts, 0.35µ
- Static 0 to 50 Megahertz
- Atomic Memory Direct Bit Manipulation of single bits
- Save and Restore of Multiple Registers
- Push and Pop of Multiple Registers
- Hardware Multiplier Unit for fast 16-bit multiplication



National Semiconductor®

REGISTER SET

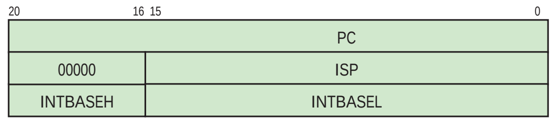
All registers are 16 bits wide, except for the four address registers, which are 21 bits wide. Bits specified as “reserved” must be written as 0, and return undefined results when read.

The internal registers of the CR16B are grouped by function:

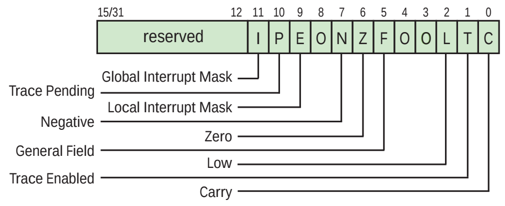
- 16 general-purpose registers
- Eight processor registers:
  - Three dedicated address registers
  - One processor status register
  - One configuration register
  - Three debug control registers

CR16B Registers

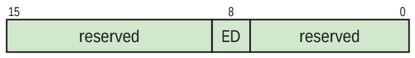
Dedicated Address Register



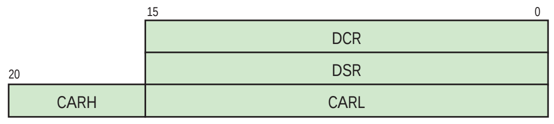
Process Status Register



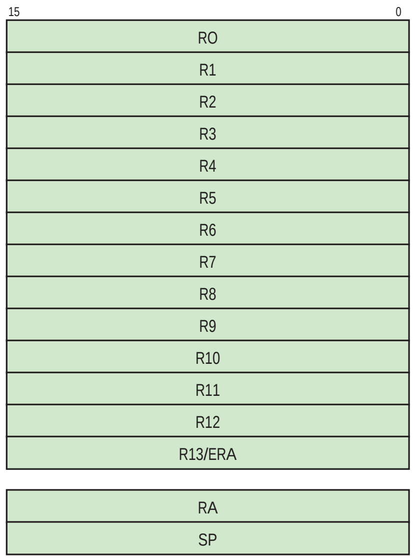
Configuration Register



Debug Registers



General Purpose Registers



CR16B SMALL AND LARGE PROGRAMMING MODELS

The CR16B has two programming modes: small and large. The small model is limited to 128 Kbytes of program address space, and 256 Kbytes of data address space. The large model supports up to 2 Mbytes of program and data space.

The two programming models are almost identical. They differ only in the instructions used for branching and program flow control.

STACKS

A stack is a one-dimensional data structure. Values are entered and removed, one item at a time, at one end of the stack called the top-of-stack. The stack consists of a block of memory, and a variable called the stack pointer. Stacks are important data structures in both systems and applications programming. They are used to store status information during sub-routine calls and interrupt servicing. In addition, algorithms for expression evaluation in compilers and interpreters use stacks to store intermediate results. High level languages, such as “C”, keep local data and other information on a stack.

The CR16B supports two kinds of stacks: the interrupt stack and the program stack.

The Interrupt Stack

The processor uses the interrupt stack to save and restore the program state during the handling of an exception condition. This information is automatically pushed, by the hardware, on to the interrupt stack before entering an exception service procedure. On exit from the exception service procedure, the hardware pops this information from the interrupt stack. The interrupt stack can reside in the first 64 Kbytes of the address range, and is accessed via the ISP processor register.

The Program Stack

The program stack is normally used by programs at run time, to save and restore register values upon procedure entry and exit. It is also used to store local and temporary variables. The program stack is accessed via the SP general-purpose register and therefore must reside in the first 64 Kbytes of the address range. Note that this stack is handled by software only, e.g., the CompactRISC “C” Compiler generates code that pushes data on to, and pops data from, the program stack. Only PUSH and POP instructions adjust the SP automatically; otherwise, software must manage the SP during save and restore operations.

Both stacks expand downward in memory, toward address zero.

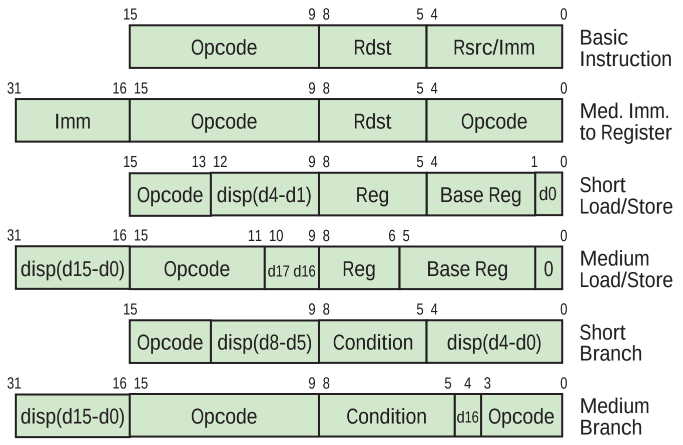
MEMORY ORGANIZATION

The CR16B implements 21-bit addresses. This allows the CPU to access up to 256 Kbytes of data, and 128 Kbytes of program memory in the small model, and 2 Mbytes of program and data in the large model. The memory is a uniform linear address space. Memory locations are numbered sequentially starting at 0 and ending at 2^{18}-1 in the small model, and at 2^{21}-1 in the large model. The number specifying a memory location is called an address.

CR16B data addressing is always byte-related (i.e., data can be addressed at byte-resolution). The instructions, by contrast, are always word-aligned, and therefore instruction addresses are always even-addressed.

INSTRUCTION SET

The following instructions are included in the CR16B



Instruction Commands

The following instructions are included in the CR16B.

Mnemonic	Operands	Description
MOVES		
MOVi	Rsrc/imm, Rdest	Move
MOVXB	Rsrc, Rdest	Move with sign extension
MOVZB	Rsrc, Rdest	Move with zero extension
MOVD	imm, (Rdest+1, Rdest)	Move 21-bit immediate to register-pair
INTEGER ARITHMETIC		
ADD[U]i	Rsrc/imm, Rdest	Add
ADDCi	Rsrc/imm, Rdest	Add with carry
MULi	Rsrc/imm, Rdest	Multiply: Rdest(8):= Rdest(8) * Rsrc(8)/Imm Rdest(16):= Rdest(16) * Rsrc(16)/Imm
MULSB	Rsrc, Rdest	Multiply: Rdest(16):= Rdest(8) * Rsrc(8)
MULSW	Rsrc, Rdest	Multiply: (Rdest+1, Rdest):= Rdest(16) * Rsrc(16)
MULUW	Rsrc, Rdest	Multiply: unsigned (Rdest+1,Rdest):= Rdest(16) * Rsrc(16)
SUBi	Rsrc/imm, Rdest	Subtract (Rdest := Rdest – Rsrc)
SUBCi	Rsrc/imm, Rdest	Subtract w/carry (Rdest – Rsrc – PSR.C)
INTEGER COMPARISON		
CMPi	Rsrc/imm, Rdest	Compare (Rdest – Rsrc)
BEQ01	Rsrc, disp	Compare Rsrc to 0 and branch if EQUAL
BNE0i	Rsrc, disp	Compare Rsrc to 0 and branch if NOT-EQUAL
BEQ1i	Rsrc, disp	Compare Rsrc to 1 and branch if EQUAL
BNE1i	Rsrc, disp	Compare Rsrc to 1 and branch if NOT-EQUAL
LOGICAL AND BOOLEAN		
ANDi	Rsrc/imm, Rdest	Logical AND
ORi	Rsrc/imm, Rdest	Logical OR
Scond	Rdest	Save condition code as boolean
XORi	Rsrc/imm, Rdest	Logical exclusive OR
SHIFTS		
ASHUi	Rsrc/imm, Rdest	Arithmetic left/right shift
LSHi	Rsrc/imm, Rdest	Logical left/right shift

Instruction Commands (cont.)

BITS		
TBIT	Roffset/imm, Rsrc	Test bit
SBITi	Iposition, 0(Rbase) Iposition, disp16(Rbase) Iposition, abs	Set a bit in memory
CBITi	Iposition, 0(Rbase) Iposition, disp16(Rbase) Iposition, abs	Clear a bit in memory
TBITi	Iposition, 0(Rbase) Iposition, disp16(Rbase) Iposition, abs	Test a bit in memory
PROCESSOR REGISTER MANIPULATION		
LPR	Rsrc, Rproc	Load processor register
SPR	Rproc, Rdest	Store processor register
JUMPS AND LINKAGE		
Bcond	disp	Conditional branch
BAL	Rlink, disp	Branch and link
BR	disp	Branch
EXCP	vector	Trap (vector)
Jcond	Rtarget	Conditional Jump
JAL	Rlink, Rtarget	Jump and link
JUMP	Rtarget	Jump
RETX		Return from exception
PUSH	imm, Rsrc	Push “imm” number of registers on user stack starting with RSRC
POP	imm, Rdest	Push “imm” number of registers on user stack starting with Rdest
POPRET	imm, Rdest	Restore registers (similar to POP) and perform JUMP RA or JUMP (RA, ERA) depending on memory model
LOAD AND STORE		
LOADi	disp(Rbase), Rdest abs, Rdest disp(Rpair+1, Rpair), Rdest	Load (register relative) Load (absolute) Load (far-relative)
STORi	Rsrc, disp(Rbase) Rsrc, disp(Rpair +1, Rpair) Rsrc, abs sm_imm, 0(Rbase) sm_imm, disp(Rbase) sm_imm, abs	Store (register relative) Store (far-relative) Store (absolute) Store small immediate in memory
LOADM	imm	Load 1 to 4 registers (R2-R5) from memory, starting at the address in R0, according to imm count value
STORM	imm	Store 1 to 4 registers (R2-R5) into memory, starting at the address in R1, according to imm count value
MISCELLANEOUS		
DI		Disable maskable interrupts
EI		Enable maskable interrupts
NOP		No operation
WAIT		Wait for interrupt
EIWAIT		Enable interrupts and wait for interrupt

©1997 NATIONAL SEMICONDUCTOR, and are registered trademarks and CompactRISC is a trademark of National Semiconductor Corporation. All rights reserved.
633170-001