Motorola DSP Products

DSP96002

32/96 Bit IEEE Floating Point
General Purpose
Digital Signal Processor

TERRY SCHULTZ
MOTOROLA
P.O. BOX 193
MESA, AZ 85201-0193

DSP96002 APPLICATIONS

- Engineering Workstations
- Personal Computers
- Graphics Processing
- Image Processing
- Numeric / Array Processing
- Laser Printers/PostScript Processing
- Radar / Sonar Processing
- Spectral Analysis
- Medical Equipment
- Digital Audio
- High Speed Control
- Speech Processing
- Instrumentation
DSP96002 Port A/B Applications

- **Graphics Processor**
  * Dual buses and use of DRAM page mode achieve a 5X performance improvement over a single bus system without DRAM support.

```
Program DRAM → DSP96002 → Graphics Data DRAM
```

- **FFT Array Processor**
  * Data bandwidth doubles with FFT code in internal RAM.

```
Real Data RAM → DSP96002 → Imag Data RAM
```

- **Simulation Engine**
  * Large caches reduce system bus loading.

```
Program Cache → DSP96002 → Data Cache
```

IEEE Floating Point DSP96002
DSP96002 Multiprocessor Support

The DSP96002 bus represents a significant advancement in:

- Multiprocessor systems and communications
- DRAM/VRAM support
- Interfacing to 32 bit microprocessors

DSP96002 user benefits include:

- Higher performance, more flexibility
- Fewer support chips, smaller footprint
- Lowest system cost

---

IEEE Floating Point DSP96002
DSP96002 DRAM/VRAM Systems

- Two Independent DMA Channels
- Generic Data Transfers
  * Memory to Memory, Including Memory Mapped Peripherals
  * Handshake Modes for Data Throttling and Peripherals
  * Internal and/or External
  * Port A and/or B sources and destinations.
- Linear, Modulo or Bit Reversed DMA Addressing
  * Shares Addressing Units with CPU
  * Separate R, N and M registers for Source and Destination (Each Channel)
- High Performance in Parallel with the CPU
  * Transfer Rates up to 54 Mbytes per Second
  * Port to Port Transfers Transparent to the CPU
  * DMA Wait States Do Not Slow Down the CPU
- Dual Access Internal Memories
  * P, X and Y RAM and ROM - DMA Access Transparent to the CPU
  * One CPU Time Slot per Instruction Cycle
  * One DMA Time Slot per Instruction Cycle
- Programmable Bus Access Priority
  * If Both CPU and DMA want the Same External Bus
  * If Both CPU and DMA want the Global Data Bus
### MOTOROLA Digital Signal Processors

#### DSP96002 User Signal Programming Model

**DSP96002 Floating Point or Integer Registers**

| DATA UNIT | D31 | D30 | D29 | D28 | D27 | D26 | D25 | D24 | D23 | D22 | D21 | D20 | D19 | D18 | D17 | D16 | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|-----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|

**Address Generation Unit**

<table>
<thead>
<tr>
<th>M7</th>
<th>M6</th>
<th>M5</th>
<th>M4</th>
<th>M3</th>
<th>M2</th>
<th>M1</th>
<th>M0</th>
</tr>
</thead>
<tbody>
<tr>
<td>N7</td>
<td>N6</td>
<td>N5</td>
<td>N4</td>
<td>N3</td>
<td>N2</td>
<td>N1</td>
<td>N0</td>
</tr>
</tbody>
</table>

**Program Control Unit**

<table>
<thead>
<tr>
<th>PC</th>
<th>MR</th>
<th>IER</th>
<th>ER</th>
<th>CCR</th>
<th>SR</th>
</tr>
</thead>
</table>

**System Stack (SS)**

<table>
<thead>
<tr>
<th>SSH</th>
<th>SSL</th>
</tr>
</thead>
</table>

---

### MOTOROLA Digital Signal Processors

#### DSP96002 Status Register Format

**CCR - Condition Code Register**

- Carry
- Overflow
- Zero
- Negative
- Infinity
- Local Reject
- Reject
- Accept

**ER - Exception Register**

- Inexact
- Divide-by-Zero
- Underflow
- Overflow
- Operation Error
- Signals NaN
- Not A Number
- Unordered Condition

**MR - Mode Register**

- Reserved
- MUL/div Precision
- Flush to Zero
- Interrupt Mask
- Reserved
- Loop Flag

**Notes:**
- Number in parentheses indicates status after power-on reset.
- *Reserved* fields should be written with zero for future compatibility.

---

*IEEE Floating Point DSP96002*
DSP96002 Operating Mode Register

DSP96002 Bootstrap Modes

- MA, MB and MC mode bits are latched from MODA, MODB and MODC pins when RESET is negated.
- If MC=1, an internal self-bootstrap program begins executing from the bootstrap ROM.
- MA and MB are read by the self-bootstrap program to select the bootstrap source (for example, an external byte-wide EPROM).
- User code is moved from the bootstrap source to the Program RAM.
- Self-bootstrap ends by jumping to the user code in Program RAM.

IEEE Floating Point DSP96002
**IEEE Floating Point Numbers**

Numbers of the form: \((-1)^s \times 2^E \times (1+b_1b_2 \ldots b_{p-1}b_p)\)

Where:
- \(s = 0\) or \(1\)
- \(E = \) any integer between \(E_{\text{min}}\) and \(E_{\text{max}}\), inclusive
- \(b_0 = 0\) or \(1\) (\(b_0\) is a hidden integer bit)

Encodings for:
- +Zero and -Zero
- +Infinity and -Infinity
- Denormalized Numbers (\(E = E_{\text{min}}\), unnormalized mantissa)
- At least one signaling Not-a-Number (NaN)
- At least one quiet NaN

**IEEE Floating Point Precisions**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Single (SP)</th>
<th>Single Extended (SEP)</th>
<th>Double (DP)</th>
<th>Double Extended (DEP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mantissa width P</td>
<td>24</td>
<td>&gt;=32</td>
<td>53</td>
<td>&gt;=64</td>
</tr>
<tr>
<td>Exponent maximum Emax</td>
<td>+127</td>
<td>&gt;=+1023</td>
<td>+1023</td>
<td>&gt;=+16383</td>
</tr>
<tr>
<td>Exponent minimum Emin</td>
<td>-126</td>
<td>&lt;=-1022</td>
<td>-1022</td>
<td>&lt;=-16382</td>
</tr>
<tr>
<td>Exponent bias Ebias</td>
<td>+127</td>
<td>Unspecified</td>
<td>+1023</td>
<td>Unspecified</td>
</tr>
<tr>
<td>Exponent width</td>
<td>8</td>
<td>&gt;=11</td>
<td>11</td>
<td>&gt;=15</td>
</tr>
<tr>
<td>Format width</td>
<td>32</td>
<td>&gt;=43</td>
<td>64</td>
<td>&gt;=79</td>
</tr>
<tr>
<td>Implementation</td>
<td>Required</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>DSP96002 Data ALU Formats</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DSP96002 Move Formats</td>
<td>Yes</td>
<td>(Yes)</td>
<td>Yes</td>
<td>(Yes)</td>
</tr>
</tbody>
</table>

**DSP96002 Data ALU Block Diagram**

- IEEE Floating Point Add/Subtract Unit
- IEEE Floating Point Multiply Unit
- Special Function Unit
  - Logical
  - Divide
  - Square Root

**Inputs/Outputs**

- XDB
- YDB

**IEE Floating Point DSP96002 Formats**

- Yes
- No
IEEE Standard Storage Formats

Bit Level Conformance Required

• SP Format (e=E+127)

```
 31 30 23 22 0
```

```
s e f
```

• DP Format (e=E+1023)

```
63 62 52 51 0
```

```
s e f
```

• SEP and DEP (extended) storage formats are not specified by IEEE and are typically not stored in memory.

**Why Use Extended Precision?**

• Greater precision and dynamic range for improved accuracy in intermediate calculations
  * Expression evaluation
  * Inner products

• SEP allows efficient computation of IEEE SP math library functions
  * Transcendentals
  * Trigonometric argument reduction
  * Power function: $Y^X$

Advantages of IEEE Floating Point

**Superior Compatibility**

• Standard storage format for data exchange with other processors.

• Numerically identical results across many implementations (coprocessors, data path chips and integrated floating point chips).

• Ease of porting high-level language software and test files using IEEE floating point.

• Interface to IEEE 754 format data bases (graphics) is easy.

**Superior Mathematical Accuracy**

• Guaranteed error bounds for add, subtract, compare, multiply, divide, remainder, square root, and conversions between fixed and floating point formats.

• Support of denormalized numbers for gradual underflow.

• Support of four rounding modes:
  * Round to nearest (even)
  * Round to zero
  * Round to plus infinity
  * Round to minus infinity

**Superior Error Handling**

• Standard floating point exception processing.

• Standard comparison predicates and relations for conditional branches.

• Support of quiet and signalling Not-a-Numbers (NaN).
Floating Point Multiplication
Single Precision Accuracy Comparison
(Sign, Exponent, Mantissa) in bits

Motorola
(IEEE Standard)

Competition
(Proprietary)

IEEE: 1 Rounding Error
Proprietary: 4 Truncation Errors

IEEE Floating Point DSP96002

Floating Point Addition/Subtraction
Single Precision Accuracy Comparison
(Sign, Exponent, Mantissa) in bits

Motorola
(IEEE Standard)

Competition
(Proprietary)

IEEE: 1 Rounding Error
Proprietary: 3 Truncation Errors

IEEE Floating Point DSP96002
DSP96002 Underflow Modes

Floating Point underflow occurs when the exponent of a normalized result is less than the format minimum. Thus, the underflowed result cannot be stored as a normalized number.

• IEEE Gradual Underflow Mode
  * The mantissa is denormalized until the exponent is equal to the format minimum.
  * If the exponent is zero, the hidden integer bit is also 0.
  * If the denormalized mantissa is all zeros, the result is floating point zero.
  * Special DSP96002 hardware detects denormalized numbers (including floating point zero) and inserts one additional instruction cycle to process each denormalized number. The additional cycles are data-dependent.

• Non-IEEE Flush to Zero Mode
  * All floating point underflows are forced to floating point zero.
  * No additional cycles (data-independent).

DSP96002 Exception Processing

• The IEEE floating point standard defines five types of exceptions:
  * Invalid Operation (0xInfinity, 0/0, Infinity/Infinity)
  * Divide by Zero
  * Overflow
  * Underflow
  * Inexact

• The IEEE standard also defines two responses for each floating point exception:
  * Trap-disabled exception processing (default)
    - Defines result to be delivered
    - Defines how status flags record the error
  * Trap-enabled exception processing (optional)
    - Defines result to be delivered
    - Defines interface to trap handler

• The DSP96002 implements IEEE trap-disabled exception processing in hardware:
  * "Sticky" status flags remember the error
  * Real-time systems must deliver a result
  * No additional cycles
DSP96002 Rounding Control

To meet IEEE error bounds, floating-point arithmetic is done to infinite precision with one final rounding error to the result precision.

- **IEEE Rounding Modes**
  - Round to Nearest (Even) (default)
  - Round to Zero
  - Round to Plus Infinity
  - Round to Minus Infinity

- **Three Instruction Classes**
  - Single Precision Rounding
    - fadd.s d1,d0
  - Single Extended Precision Rounding
    - fadd.x d1,d0
  - No Rounding
    - fcmp d1,d0

DSP96002 Opcode Summary

- **Floating-Point Operations**
  - fabs
  - fadd
  - faddsub
  - fcfr
  - fcmp
  - fcmpg
  - fcmpm
  - fcopy
  - fgetman
  - fint
  - float
  - floatu
  - floor
  - fmpy
  - fmpy // fadd
  - fmpy // faddsub
  - fmpy // fsub
  - fneg
  - fscale
  - fseedr
  - fsub
  - ftr
  - fstat

- **Integer Operations**
  - abs
  - addc
  - add
  - asl
  - asr
  - bfind
  - clr
  - cmp
  - cmpg
  - cmpm
  - dec
  - ext
  - extb
  - getexp
  - inc
  - int
  - intr
  - intu
  - inturz
  - mpy
DSP96002 Opcode Summary

• Integer Operations (continued)
  - mpyu: Unsigned Multiply
  - neg: Negate
  - negc: Negate with Carry
  - setr: Set Register
  - sub: Subtract
  - subc: Subtract with Carry
  - tfr: Transfer Register
  - tst: Test

• Logical Operations
  - and: AND
  - andc: AND with Complement
  - andi: AND with Control Register
  - btst: Bit Test
  - eor: Exclusive OR
  - join: Join Half-Words
  - joinb: Join Bytes
  - lsl: Logical Shift Left
  - lsr: Logical Shift Right
  - not: Complement
  - or: OR
  - orc: OR with Complement
  - ori: OR with Control Register
  - rol: Rotate Left
  - ror: Rotate Right
  - split: Split Half-Words
  - splitb: Split Bytes

• Bit Manipulation Operations (Read-Modify-Write)
  - bhcg: Bit Test and Change
  - bclr: Bit Test and Clear
  - bset: Bit Test and Set

• Move Operations
  - move: Move
  - movec: Move Control Register
  - movep: Move Peripheral
  - moveta: Move and Test Address Register
  - lea: Load Effective Address
  - lra: Load PC Relative Address

• Loop Control Operations
  - do: Start Hardware Loop
  - dor: Start PC Relative Hardware Loop
  - enddo: End Current Loop
  - rep: Repeat Next Instruction

• Program Control Operations
  - bcc: Branch on Condition cc
  - brcir: Branch if Bit Clear
  - brcs: Branch on Condition cc
  - bscs: Branch on Condition cc
  - bset: Branch if Set
  - bsse: Branch if Bit Set
  - fbec: Float Branch on Condition cc
  - fbes: Float Branch to Subroutine on Condition cc
  - ftraps: Float Trap on Condition cc
  - illegal: Illegal Instruction Trap
  - jcc: Jump on Condition cc
  - jcl: Jump if Bit Clear
  - jsec: Jump to Subroutine on Condition cc
  - jscc: Jump to Subroutine if Bit Clear
  - jssel: Jump if Bit Set
  - jsset: Jump to Subroutine if Bit Set
  - nop: No Operation
  - reset: Reset On-Chip Peripherals
  - rti: Return From Interrupt
  - rts: Return From Subroutine
  - trap: Stop Processing (Low Power Standby)
  - trapcc: Trap on Condition cc
  - wait: Wait for Interrupt (Low Power Standby)
**DSP9602 Parallel Operations**

**Instruction Syntax**

OPCODE (OPERANDS) (OPCODE OPERANDS) (PARALLEL MOVE) (PARALLEL MOVE)

**Single Precision Parallel Moves**

<table>
<thead>
<tr>
<th>Example</th>
<th>OPCODE (OPERANDS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>fmipy d4,d5,d0 fadd.s d6,d1</td>
</tr>
<tr>
<td>Register</td>
<td>fmipy d4,d5,d0 fadd.s d6,d1 d1.s,d4.s</td>
</tr>
<tr>
<td>Update</td>
<td>fmipy d4,d5,d0 fadd.s d6,d1 (r0)+</td>
</tr>
<tr>
<td>X memory</td>
<td>fmipy d4,d5,d0 fadd.s d6,d1 x:(r7)-,d6.s</td>
</tr>
<tr>
<td>X memory and register</td>
<td>fmipy d4,d5,d0 fadd.s d6,d1 d3.s,x:(r2)+ d7.s,d3.s</td>
</tr>
<tr>
<td>Y memory</td>
<td>fmipy d4,d5,d0 fadd.s d6,d1 d10.ly:(r0+$43)</td>
</tr>
<tr>
<td>Y memory and register</td>
<td>fmipy d4,d5,d0 fadd.s d6,d1 d2.s,d7.s y:(r6)+n6,d3.s</td>
</tr>
<tr>
<td>X and Y memory</td>
<td>fmipy d4,d5,d0 fadd.s d6,d1 x:(r3)+,d4.s d1.s,y:(r4)-</td>
</tr>
</tbody>
</table>

**Double Precision Parallel Moves**

<table>
<thead>
<tr>
<th>Example</th>
<th>OPCODE (OPERANDS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>fmipy d4,d5,d0 fadd.s d6,d1 d2.d,d3.d</td>
</tr>
<tr>
<td>L(X:Y) memory</td>
<td>fmipy d4,d5,d0 fadd.s d6,d1 d2.d,k:(r0)-n0</td>
</tr>
</tbody>
</table>

**Conditional Operations**

| Integer cc, no CCR update | fmipy d4,d5,d0 fadd.s d6,d1 ifcs |
| Float cc, no CCR update   | fmipy d4,d5,d0 fadd.s d6,d1 fflt |
| Integer cc, CCR update    | lsl d1 ifcs.u |
| Float cc, CCR Update      | ftrr.s d0,d1 figt.u |

**Double Precision Parallel Moves**

<table>
<thead>
<tr>
<th>Example</th>
<th>OPCODE (OPERANDS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>fmipy d4,d5,d0 fadd.s d6,d1 d2.d,d3.d</td>
</tr>
<tr>
<td>L(X:Y) memory</td>
<td>fmipy d4,d5,d0 fadd.s d6,d1 d2.d,l:(r0)-n0</td>
</tr>
</tbody>
</table>

**Conditional Operations**

| Integer cc, no CCR update | fmipy d4,d5,d0 fadd.s d6,d1 ifcs |
| Float cc, no CCR update   | fmipy d4,d5,d0 fadd.s d6,d1 fflt |
| Integer cc, CCR update    | lsl d1 ifcs.u |
| Float cc, CCR Update      | ftrr.s d0,d1 figt.u |
DSP96002 Parallel Operations

Instruction Syntax

<table>
<thead>
<tr>
<th>Addressing Mode Summary</th>
</tr>
</thead>
</table>

**Instruction Syntax**

OPCODE (OPERANDS) (OPCODE OPERANDS) (PARALLEL MOVE) (PARALLEL MOVE)

**Single Precision Parallel Moves**

- None: `fmpy d4,d5,d0 fadd.s d6,d1`
- Register: `fmpy d4,d5,d0 fadd.s d6,d1 d1 s,d4.s`
- Update: `fmpy d4,d5,d0 fadd.s d6,d1 (r0)+`
- X memory: `fmpy d4,d5,d0 fadd.s d6,d1 x:(r7).d6.s`
- X memory and register: `fmpy d4,d5,d0 fadd.s d6,d1 d3.s,x:(r2)+ d7.s,d3.s`
- Y memory: `fmpy d4,d5,d0 fadd.s d6,d1 d0 l.y:(r0+$43)`
- Y memory and register: `fmpy d4,d5,d0 fadd.s d6,d1 x:(r3)+,d4.s d1 s,y:(r4)-`
- X and Y memory: `fmpy d4,d5,d0 fadd.s d6,d1 x:(r5),d4.s y,d5.s`

**Double Precision Parallel Moves**

- Register: `fmpy d4,d5,d0 fadd.s d6,d1 d2 d3.d`
- L(X:Y) memory: `fmpy d4,d5,d0 fadd.s d6,d1 d2 d1.(r0)-n0`

**Conditional Operations**

- Integer cc, no CCR update: `fmpy d4,d5,d0 fadd.s d6,d1 ifcs`
- Float cc, no CCR update: `fmpy d4,d5,d0 fadd.s d6,d1 fflt`
- Integer cc, CCR update: `lsl d1 ifcs.u`
- Float cc, CCR Update: `fltr.s d0,d1 fflt.u`

**Addressing Mode**

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Modifiers</th>
<th>Assembler Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register Direct</td>
<td>Low data register</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Middle data register</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>High data register</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Single precision integer</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Double precision integer</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Single precision float</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Double precision float</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>Address or Control</td>
<td>no</td>
</tr>
</tbody>
</table>

**Address Register Indirect**

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Instruction Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td>(Rn)</td>
</tr>
<tr>
<td>Post-increment by 1</td>
<td>(Rn)+</td>
</tr>
<tr>
<td>Post-decrement by 1</td>
<td>(Rn)-</td>
</tr>
<tr>
<td>Post-increment by offset Nn</td>
<td>(Rn)+Nn</td>
</tr>
<tr>
<td>Post-decrement by offset Nn</td>
<td>(Rn)-Nn</td>
</tr>
<tr>
<td>Indexed by offset Nn</td>
<td>(Rn)+Nn</td>
</tr>
<tr>
<td>Indexed by displacement</td>
<td>(Rn)+expr</td>
</tr>
<tr>
<td>Pre-decrement by 1</td>
<td>(Rn)-</td>
</tr>
</tbody>
</table>

**PC Relative Indirect**

<table>
<thead>
<tr>
<th>Offset Length</th>
<th>Instruction Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 bit short displacement</td>
<td>no</td>
</tr>
<tr>
<td>32 bit long displacement</td>
<td>no</td>
</tr>
<tr>
<td>Address register displacement</td>
<td>no</td>
</tr>
</tbody>
</table>

**Special**

<table>
<thead>
<tr>
<th>Offset Length</th>
<th>Instruction Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bit immediate short data</td>
<td>no</td>
</tr>
<tr>
<td>32 bit immediate long data</td>
<td>no</td>
</tr>
<tr>
<td>8 bit absolute short address</td>
<td>no</td>
</tr>
<tr>
<td>12 bit short jump address</td>
<td>no</td>
</tr>
<tr>
<td>32 bit absolute long address</td>
<td>no</td>
</tr>
<tr>
<td>Implicit</td>
<td>no</td>
</tr>
</tbody>
</table>

**Where**

- Dn is data register n = 0-9
- Rn is address register n = 0-7
- Nn is offset register n = 0-7
- expr is any valid assembler expression
Displacement Addressing Mode

- Assembler Syntax: \( (Rn + SAAAAAAA) \)
- Operands referenced: \( P, X, Y, L \) memories.
- Additional instruction execution time (clocks): 4
- Additional effective address words: 1

Example: MOVE d0,s,X,(R6+$4)

Before Execution

<table>
<thead>
<tr>
<th>X Memory</th>
<th>X Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$6004</td>
<td>$6004</td>
</tr>
<tr>
<td>$6000</td>
<td>$6000</td>
</tr>
</tbody>
</table>

R6 00000000  31
N6 XXXXXXXX  21
M6 FFFFFFFF  11

After Execution

<table>
<thead>
<tr>
<th>X Memory</th>
<th>X Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$6004</td>
<td>$6004</td>
</tr>
<tr>
<td>$6000</td>
<td>$6000</td>
</tr>
</tbody>
</table>

R6 00000000  31
N6 XXXXXXXX  21
M6 FFFFFFFF  11

PC Relative Addressing Mode - Branches

- Assembler Syntax: BRA offset
- Operands referenced: \( P \) Memory
- Additional instruction execution time (clocks): 4 (15 bit signed offset)
- Additional effective address words: 0 (15 bit signed offset)

Example: BRA LABEL1

Before Execution

<table>
<thead>
<tr>
<th>Branch Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>$04107000</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>Nest Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$04107004</td>
</tr>
</tbody>
</table>

Before Execution

<table>
<thead>
<tr>
<th>P Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$04107000</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>P Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$04107000</td>
</tr>
</tbody>
</table>

Example: BRA LABEL1

Before Execution

<table>
<thead>
<tr>
<th>P Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$04107000</td>
</tr>
</tbody>
</table>

After Execution

<table>
<thead>
<tr>
<th>Nest Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$04107004</td>
</tr>
</tbody>
</table>
PC Relative Addressing Mode - Load Relative Address

- Assembler Syntax:
  LRA (PC + $AAAAAAA), reg ; static offset
  LRA (PC + Rn), reg ; dynamic offset
- Additional instruction execution time (clocks): 4 Static, 2 Dynamic
- Additional effective address words: 1 static, 0 dynamic

**Example:**

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00000100</td>
<td>LRA (PC+$150), d4.1</td>
</tr>
<tr>
<td>$00000101</td>
<td>$00000150</td>
</tr>
<tr>
<td>$00000102</td>
<td>xxxxxxx</td>
</tr>
</tbody>
</table>

\[ d4.1 = \$102 + \$150 = \$252 \]

**Example:**

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00000100</td>
<td>LRA (PC+r1), r4</td>
</tr>
<tr>
<td>$00000101</td>
<td>xxxxxxx</td>
</tr>
<tr>
<td>$00000102</td>
<td>xxxxxxx</td>
</tr>
</tbody>
</table>

\[ r1 = \$150 \]

\[ r4 = \$101 + \$150 = \$251 \]
Multiple Wraparound Modulo $2^M$ Addressing

1. Modifier Register $Mn = \$FFmm mmmm$, where $mm mmmm = \text{bit mask}$. $XX...XX00...00$ where $M = \text{number of Isb's set in} Mn$

2. Lower Bound $= XX...XX11...11$

3. Upper Bound $= XX...XX00...00$

4. Lower Bound $\leq$ Address Register $Rn \leq$ Upper Bound

5. Offset Register $Nn = \text{any value}$

Example: $\text{MOVE} d0.s,X:(R0)+N0$

Starting Address $R0 = 165$

Increment $N0 = 271$

Modulo $M0 = \$FF00003F$

Applications: Waveform Generation, Modulation

Pipeline Timing For One DSP96002 Instruction

Instruction:

- $\text{fmpy d4,d5,d0}$
- $\text{faddsub.s d6,d1, x:(r0)+n0,d4.s}$
- $\text{d5.s,y:(r5)-}$

Clock Phase:

- $\text{PDB}$
  - Program Controller
  - Dual Address Generation Unit
  - Data Unit
  - Data Unit
  - Data Unit
  - $XDB$
  - $YDB$
  - $DDB$
  - $GDB$
  - Port A Bus
  - Port B Bus

Cache Memory:

- Modulus
- Modulus
- Modulus

Number of Instruction Words: 1

Execution Time: 50 ns

Peak Floating Point: 60 MFLOPS

Millions of Addresses per Second: 80

Mbytes per Second DMA Data Transfer: 80

IEEE Floating Point DSP96002
### DSP96002 Benchmark Summary

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Instruction</th>
<th>Cycles</th>
<th>Execution Time (40 MHz, 1990)</th>
<th>MIPS</th>
<th>SPSEP MFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Real</strong></td>
<td>v = v + s</td>
<td>2N</td>
<td>-</td>
<td>20.0</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>v = v + v</td>
<td>2N</td>
<td>-</td>
<td>20.0</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>v = v * v</td>
<td>2N</td>
<td>-</td>
<td>20.0</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>v = v + s + v</td>
<td>2N</td>
<td>-</td>
<td>20.0</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>v = v + v + s</td>
<td>2N</td>
<td>-</td>
<td>20.0</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>v = v * v + v</td>
<td>2N</td>
<td>-</td>
<td>20.0</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>Polynomial Expansion</td>
<td>2N</td>
<td>-</td>
<td>20.0</td>
<td>29.9</td>
</tr>
<tr>
<td></td>
<td>Convolution or Correlation</td>
<td>1N</td>
<td>-</td>
<td>20.0</td>
<td>40.0</td>
</tr>
<tr>
<td></td>
<td>FIR Filter With Data Shift</td>
<td>1N</td>
<td>50 ns/tap</td>
<td>20.0</td>
<td>40.0</td>
</tr>
<tr>
<td></td>
<td>Biquad Filter With Data Shift</td>
<td>4N</td>
<td>200 ns/stage</td>
<td>20.0</td>
<td>40.0</td>
</tr>
<tr>
<td></td>
<td>Lattice Filter With Data Shift</td>
<td>3N</td>
<td>150 ns/stage</td>
<td>20.0</td>
<td>26.7</td>
</tr>
<tr>
<td></td>
<td>[10x10][10x10] = [10x10]</td>
<td>1583</td>
<td>79 us</td>
<td>18.5</td>
<td>29.0</td>
</tr>
<tr>
<td></td>
<td>1024 Point FFT With Bit Reversal</td>
<td>12880</td>
<td>644 us</td>
<td>19.1</td>
<td>35.0</td>
</tr>
<tr>
<td></td>
<td><strong>Complex</strong></td>
<td>v = v * v</td>
<td>4N</td>
<td>-</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>v = v * v + v</td>
<td>4N</td>
<td>-</td>
<td>20.0</td>
<td>40.0</td>
</tr>
<tr>
<td></td>
<td>Convolution or Correlation</td>
<td>4N</td>
<td>-</td>
<td>20.0</td>
<td>40.0</td>
</tr>
<tr>
<td></td>
<td>FIR Filter With Data Shift</td>
<td>4N</td>
<td>200 ns/tap</td>
<td>20.0</td>
<td>40.0</td>
</tr>
<tr>
<td></td>
<td>1024 Point FFT With Bit Reversal</td>
<td>20931</td>
<td>944 us</td>
<td>19.8</td>
<td>43.1</td>
</tr>
<tr>
<td></td>
<td><strong>Graphic/Image Processing</strong></td>
<td>Divide</td>
<td>32 Bit Accuracy</td>
<td>7</td>
<td>0.30 us</td>
</tr>
<tr>
<td></td>
<td><strong>Graphic/Image Processing</strong></td>
<td>Divide</td>
<td>16 Bit Accuracy</td>
<td>5</td>
<td>0.20 us</td>
</tr>
<tr>
<td></td>
<td><strong>Graphic/Image Processing</strong></td>
<td>Divide</td>
<td>8 Bit Accuracy</td>
<td>2</td>
<td>0.10 us</td>
</tr>
<tr>
<td></td>
<td>Square Root</td>
<td>32 Bit Accuracy</td>
<td>12</td>
<td>0.50 us</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>Square Root</td>
<td>16 Bit Accuracy</td>
<td>8</td>
<td>0.30 us</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>Square Root</td>
<td>8 Bit Accuracy</td>
<td>2</td>
<td>0.10 us</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>Transformations</td>
<td>[1x3][3x3] = [1x3]</td>
<td>12</td>
<td>0.60 us</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>Transformations</td>
<td>[3x3][3x3] = [3x3]</td>
<td>30</td>
<td>1.50 us</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>Transformations</td>
<td>[1x4][4x4] = [1x4]</td>
<td>19</td>
<td>0.85 us</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>Transformations</td>
<td>[4x4][4x4] = [4x4]</td>
<td>67</td>
<td>3.35 us</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>3D Volume Compare For Trivial Accept/Reject</td>
<td>1 Point</td>
<td>8</td>
<td>0.40 us</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>3D Volume Compare For Trivial Accept/Reject</td>
<td>2 Point Polyline</td>
<td>14</td>
<td>0.70 us</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>3D Volume Compare For Trivial Accept/Reject</td>
<td>N Point Polygon</td>
<td>6N</td>
<td>-</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>Bezier Cubic Evaluation For Font Compilation</td>
<td>13</td>
<td>0.65 us</td>
<td>20.0</td>
<td>21.5</td>
</tr>
<tr>
<td></td>
<td>BITBLT (Bit Block Transfer), N 32 Bit Words, Replacement</td>
<td>3N</td>
<td>213 Mbits/sec</td>
<td>20.0</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>2D [3x3] Image Convolution</td>
<td>2N</td>
<td>-</td>
<td>20.0</td>
<td>37.8</td>
</tr>
</tbody>
</table>

s = scalar, v = vector, n=number of data points

---

IEEE Floating Point DSP96002