CS252
Graduate Computer Architecture

Lecture 14:
Instruction Set #1: RISC/MIPS and DSPs

March 9, 2001
Prof. David A. Patterson
Computer Science 252
Spring 2001

Multiprocessor Review

- Some optimism about future
  - Parallel processing beginning to be understood in some domains
  - More performance than that achieved with a single-chip microprocessor
  - MPs are highly effective for multiprogrammed workloads
  - MPs proved effective for intensive commercial workloads, such as OLTP (assuming enough I/O to be CPU-limited), DSS applications (where query optimization is critical), and large-scale, web searching applications
- On-chip MPs appears to be growing
  - 1) embedded market where natural parallelism often exists an obvious alternative to faster less silicon efficient, CPU. 2) diminishing returns in high-end microprocessor encourage designers to pursue on-chip multiprocessing

3 Targets of Instruction Set

- Desktop computing: performance of programs with integer and floating point data types, little regard for program size or processor power
  - Code size never reported in 4 generations of SPEC benchmarks
- Servers: today primarily for database, file server, and web applications
  - FP performance << integers and character string performance
- Embedded applications: value cost and power, so code size important because less memory cheaper and lower power
  - Embedded MPU less die area than on-chip instruction memory!

4 Classes of Instructions

Comparing Number of Instructions

° Code sequence for C = A + B for four classes of instruction sets:

Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R1,B	Load R2,B
Add	Store C	Store C, R1	Add R3,R1,R2
Pop C			Store C,R3

- Moore's Law+ graph coloring register allocator algorithm => all machines use registers today
  - Only exception: Java Virtual Machine, intended as a SW interpreter, but some have made HW version
  - Some DSPs have an accumulator (Multiply-Accumulate)

Memory Operands per Instruction

Number of memory addresses	Maximum number of operands allowed	Examples
0	3	Alpha, ARM, MIPS, PowerPC, SPARC, SuperH, Trimedia CPU64
1	2	Intel 80x86, Motorola 68000, TI TMS320C54x
2	2	VAX (also has 3-operand formats)
3	3	VAX (also has 2-operand formats)

- RISC machines tend to have only register operands + Load/Store

Numerous addressing modes has been tried

Addressing mode	Example	Meaning
Register	Add R4,R3	R4← R4+R3
Immediate	Add R4,#3	R4 ← R4+3
Displacement	Add R4,100(R1)	R4 ← R4+Mem[100+R1]
Register indirect	Add R4,(R1)	R4 ← R4+Mem[R1]
Indexed / Base	Add R3,(R1+R2)	R3 ← R3+Mem[R1+R2]
Direct or absolute	Add R1,(1001)	R1 ← R1+Mem[1001]
Memory indirect	Add R1,@(R3)	R1 ← R1+Mem[Mem[R3]]
Auto-increment	Add R1,(R2)+	R1 ← R1+Mem[R2]; R2 ← R2+d
Auto-decrement	Add R1,–(R2)	R2 ← R2–d; R1 ← R1+Mem[R2]
Scaled	Add R1,100(R2)[R3]	R1 ← R1+Mem[100+R2+R3*d]

Why Auto-increment/decrement? Scaled?

Addressing Mode Usage? (ignore register mode)

3 desktop programs:

•Displacement:	42% avg, 32% to 55%
•Immediate:	33% avg, 17% to 43%
•Register deferred (indirect):	13% avg, 3% to 24%
•Scaled:	7% avg, 0% to 16%
•Memory indirect:	3% avg, 1% to 6%
•Misc:	2% avg, 0% to 3%

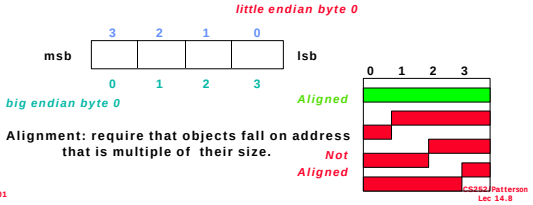
75%

88%

75% displacement & immediate
88% displacement, immediate & register indirect

Addressing and Alignment: how do byte addresses map onto words? restrictions?

- Big Endian: address of most significant IBM 370, Motorola 68k, MIPS, Sparc, HP
- Little Endian: address of least significant Intel 80x86



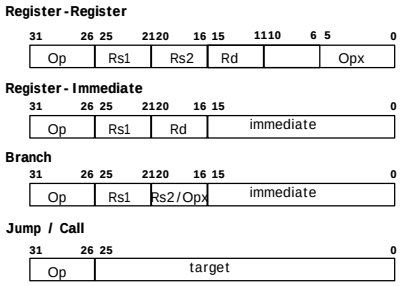
Alignment: require that objects fall on address that is multiple of their size.

A "Typical" RISC

- 32-bit fixed format instruction (3 formats)
- Memory access only via load/store instrutions
- 32 32-bit GPR (R0 contains zero, DP take pair)
  - 64-bit addresses => 64-bit registers; all desktop RISCs today
- 3-address, reg-reg arithmetic instruction; registers in same place
- Single address mode for load/store: base + displacement
  - no indirection (except via registers)
- Simple branch conditions
- Delayed branch

see: SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC, CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

Example: MIPS (Note register location)



MIPS arithmetic instructions

Instruction	*Example*	*Meaning*	*Comments*
add	add $1,$2,$3	$1 = $2 + $3	3 operands; exception possible
subtract	sub $1,$2,$3	$1 = $2 – $3	3 operands; exception possible
add immediate	addi $1,$2,100	$1 = $2 + 100	+ constant; exception possible
add unsigned	addu $1,$2,$3	$1 = $2 + $3	3 operands; no exceptions
subtract unsigned	subu $1,$2,$3	$1 = $2 – $3	3 operands; no exceptions
add imm. unsign.	addiu $1,$2,100	$1 = $2 + 100	+ constant; no exceptions
multiply	mult $2,$3	Hi, Lo = $2 x $3	64-bit signed product
multiply unsigned	multu$2,$3	Hi, Lo = $2 x $3	64-bit unsigned product
divide	div $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Lo = quotient, Hi = remainder
divide unsigned	divu $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Unsigned quotient & remainder
Move from Hi	mfhi $1	$1 = Hi	Used to get copy of Hi
Move from Lo	mflo $1	$1 = Lo	Used to get copy of Lo

MIPS logical instructions

Instruction	*Example*	*Meaning*	*Comments*
and	and $1,$2,$3	$1 = $2 & $3	3 reg. operands; Logical AND
or	or $1,$2,$3	$1 = $2 | $3	3 reg. operands; Logical OR
xor	xor $1,$2,$3	$1 = $2 ⊕ $3	3 reg. operands; Logical XOR
nor	nor $1,$2,$3	$1 = ~($2 |$3)	3 reg. operands; Logical NOR
and immediate	andi $1,$2,10	$1 = $2 & 10	Logical AND reg, constant
or immediate	ori $1,$2,10	$1 = $2 | 10	Logical OR reg, constant
xor immediate	xori $1, $2,10	$1 = ~$2 &~10	Logical XOR reg, constant
shift left logical	sll $1,$2,10	$1 = $2 << 10	Shift left by constant
shift right logical	srl $1,$2,10	$1 = $2 >> 10	Shift right by constant
shift right arithm.	sra $1,$2,10	$1 = $2 >> 10	Shift right (sign extend)
shift left logical	sllv $1,$2,$3	$1 = $2 << $3	Shift left by variable
shift right logical	srlv $1,$2, $3	$1 = $2 >> $3	Shift right by variable
shift right arithm.	srav $1,$2, $3	$1 = $2 >> $3	Shift right arith. by variable

MIPS data transfer instructions

Instruction	Comment
SW 500(R4), R3	Store word
SH 502(R2), R3	Store half
SB 41(R3), R2	Store byte
LW R1, 30(R2)	Load word
LH R1, 40(R3)	Load halfword
LHU R1, 40(R3)	Load halfword unsigned
LB R1, 40(R3)	Load byte
LBU R1, 40(R3)	Load byte unsigned
LUI R1, 40	Load Upper Immediate (16 bits shifted left by 16)

Why need LUI?

MIPS jump, branch, compare instructions

Instruction	Example	Meaning
branch on equal	beq $1,$2,100	if ($1 == $2) go to PC+4+100
	Equal test; PC relative branch	
branch on not eq.	bne $1,$2,100	if ($1!= $2) go to PC+4+100
	Not equal test; PC relative	
set on less than	slt $1,$2,$3	if ($2 < $3) $1=1; else $1=0
	Compare less than; 2's comp.	
set less than imm.	slti $1,$2,100	if ($2 < 100) $1=1; else $1=0
	Compare < constant; 2's comp.	
set less than uns.	sltu $1,$2,$3	if ($2 < $3) $1=1; else $1=0
	Compare less than; natural numbers	
set l. t. imm. uns.	sltiu $1,$2,100	if ($2 < 100) $1=1; else $1=0
	Compare < constant; natural numbers	
jump	j 10000	go to 10000
	Jump to target address	
jump register	jr $31	go to $31
	For switch, procedure return	
jump and link	jal 10000	$31 = PC + 4; go to 10000
	For procedure call	

CS 252 Administrivia

- Quiz #1 Wed March 7 5:30-8:30 306 Soda
- Pizza at LaVal's
- Cal-Stanford Day?
- Next Wednesday Christoforos Kozyrakis lecture on multimedia and vector instruction sets
  - He is leading the design of a 100M+ transistor microprocessor at Berkeley which is a vector microprocessor for multimedia applications

Pitfall: Innovating at the instruction set architecture to reduce code size without accounting for the compiler.

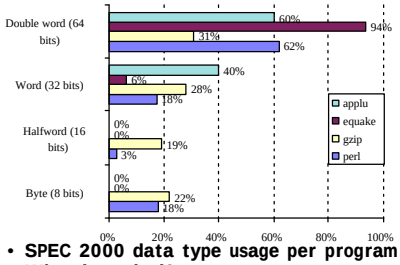
Compiler	Green Hills: Multi2000 Version 2.0	Algorithmics SDE 40B	IDT/c 7.2.1
Auto Correlation	2.1	1.1	2.7
Convolutional Encoder	1.9	1.2	2.4
Fixed-point Bit Allocation	2.0	1.2	2.3
Fixed Point Complex FFT	1.1	2.7	1.8
Viterbi GSM Decoder	1.7	0.8	1.1
Geometric Mean	1.7	1.4	2.0

- Relative MIPS Code size on EEMBC Telecom benchmarks vs. Apogee Software Version 4.1 C compiler

Pitfall: Designing a "high-level" instruction set feature specifically oriented to supporting a HLL structure

- For example, VAX CALLS instruction steps
  1) Align the stack if needed
  2) Push argument count on the stack
  3) Save registers indicated by call mask on the stack
  4) Push the return address, top and base of stack pointers on the stack
  5) Clear the condition codes
  6) Push status word and a zero word on the stack.
  7) Update the two stack pointers.
  8) Branch to the first instruction of the procedure.
- Architecture overkill: procs know # args, faster linkage: CALLS slow, mismatch

Fallacy: There is such a thing as a typical program



- SPEC 2000 data type usage per program. What is typical?

DSPs and Media processors

- Both Typically embedded applications
- Difference is real-time performance, data I/O
  - Worst case performance vs. average case performance
  - Infinite, continuous streams of data vs. fixed data set
- Small number of key kernels critical, often supplied by manufacturer
  - Libraries are important, widely used
  - Include tricks to improve performance for targeted kernels but no compiler will generate

DSP Introduction

- Digital Signal Processing: application of mathematical operations to digitally represented signals
- Signals represented digitally as sequences of samples
- Digital signals obtained from physical signals via tranducers (e.g., microphones) and analog-to-digital converters (ADC)
- Digital signals converted back to physical signals via digital-to-analog converters (DAC)
- Digital Signal Processor (DSP): electronic system that processes digital signals

Common DSP algorithms and applications

- Applications – Instrumentation and measurement
  - Communications
  - Audio and video processing
  - Graphics, image enhancement, 3- D rendering
  - Navigation, radar, GPS
  - Control - robotics, machine vision, guidance
- Algorithms
  - Frequency domain filtering - FIR and IIR
  - Frequency- time transformations - FFT
  - Correlation

What Do DSPs Need to Do Well?

- Most DSP tasks require:
  - Repetitive numeric computations
  - Attention to numeric fidelity
  - High memory bandwidth, mostly via array accesses
  - Real-time processing
- DSPs must perform these tasks efficiently while minimizing:
  - Cost
  - Power
  - Memory use
  - Development time

Who Cares about DSPs?

- DSP is a key enabling technology for many types of electronic products
- DSP-intensive tasks are the performance bottleneck in many computer applications today
- Computational demands of DSP-intensive tasks are increasing very rapidly
- In many embedded applications, general-purpose microprocessors are not competitive with DSP-oriented processors today
- 1997 market for DSP processors: $3 billion
- Texas Instruments sold off other divisions, is a DSP company today

A Tale of Two Cultures

- General Purpose Microprocessor traces roots back to Eckert, Mauchly, Von Neumann (ENIAC)
- DSP evolved from Analog Signal Processors, using analog hardware to transform phyical signals (classical electrical engineering)
- ASP to DSP because
  - DSP insensitive to environment (e.g., same response in snow or desert if it works at all)
  - DSP performance identical even with variations in components; 2 analog systems behavior varies even if built with same components with 1% variation
- Different history and different applications led to different terms, different metrics, some new inventions
- Increasing markets leading to cultural warfare

DSP vs. General Purpose MPU

- DSPs tend to be written for 1 program, not many programs.
  - Hence OSes are much simpler, there is no virtual memory or protection, ...
- DSPs sometimes run hard real-time apps
  - You must account for anything that could happen in a time slot
  - All possible interrupts or exceptions must be accounted for and their collective time be subtracted from the time interval.
  - Therefore, exceptions are BAD!
- DSPs have an infinite continuous data stream

Today's DSP "Killer Apps"

- In terms of dollar volume, the biggest markets for DSP processors today include:
  - Digital cellular telephony
  - Pagers and other wireless systems
  - Modems
  - Disk drive servo control
- Most demand good performance
- All demand low cost
- Many demand high energy efficiency
- Trends are towards better support for these (and similar) major applications.

DSP Assumptions of the World

- Machines issue/execute/complete in order
- Machines issue 1 instruction per clock
- Each line of assembly code = 1 instruction
- Clocks per Instruction = 1.000
- Floating Point is slow, expensive

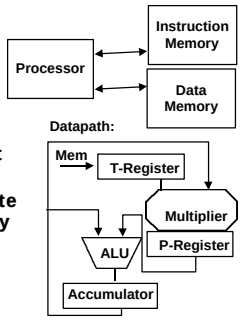
FIR filter on (simple) General Purpose Processor

```
loop:
  lw   x0, 0(r0)
  lw   y0, 0(r1)
  mul a, x0,y0
  add       y0,a,b
  sw   y0,(r2)
  inc  r0
  inc  r1
  inc  r2
  dec ctr
  tst ctr
  jnz loop
```

- Problems: Bus / memory bandwidth bottleneck, control code overhead

First Generation DSP (1982): Texas Instruments TMS32010

- 16-bit fixed-point
- "Harvard architecture"
  - separate instruction, data memories
- Accumulator
- Specialized instruction set
  - Load and Accumulate
- 390 ns Multiple-Accumulate (MAC) time; 228 ns today



TMS32010 FIR Filter Code

- Here X4, H4, ... are direct (absolute) memory addresses:

```
LT X4   ; Load T with x(n-4)
MPY H4  ; P = H4*X4
LTD X3  ; Load T with x(n-3); x(n-4) = x(n-3);
  Acc = Acc + P
MPY H3  ; P = H3*X3
LTD X2
MPY H2
...
```

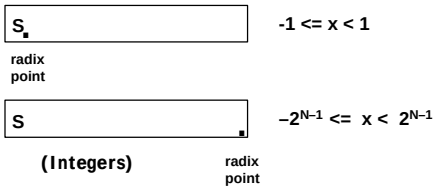
- Two instructions per tap, but requires unrolling

Features Common to Most DSP Processors

- Data path configured for DSP
- Specialized instruction set
- Multiple memory banks and buses
- Specialized addressing modes
- Specialized execution control
- Specialized peripherals for DSP

DSP Data Path: Arithmetic

- DSPs dealing with numbers representing real world => Want "reals"/ fractions
- DSPs dealing with numbers for addresses => Want integers
- Support "fixed point" as well as integers



DSP Data Path: Precision

- Word size affects precision of fixed point numbers
- DSPs have 16-bit, 20-bit, or 24-bit data words
- Floating Point DSPs cost 2X - 4X vs. fixed point, slower than fixed point
- DSP programmers will scale values inside code
  - SW Libraries
  - Seperate explicit exponent
- "Blocked Floating Point" single exponent for a group of fractions
- Floating point support simplify development

DSP Data Path: Overflow?

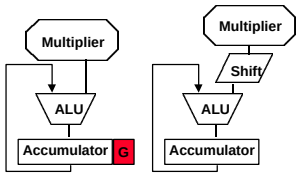
- DSP are descended from analog : what should happen to output when "peg" an input? (e.g., turn up volume control knob on stereo)
  - Modulo Arithmetic???
- Set to most positive ($2^{N-1}-1$) or most negative value(-2^{N-1}) : "saturation"
- Many algorithms were developed in this model

DSP Data Path: Multiplier

- Specialized hardware performs all key arithmetic operations in 1 cycle
- ~ 50% of instructions can involve multiplier => single cycle latency multiplier
- Need to perform multiply-accumulate (MAC)
- n-bit multiplier => 2n-bit product

DSP Data Path: Accumulator

- Don't want overflow or have to scale accumulator
- Option 1: accumalator wider than product: "guard bits"
  - Motorola DSP: 24b x 24b => 48b product, 56b Accumulator
- Option 2: shift right and round product before adder



DSP Data Path: Rounding for Fixed Pt.

- Even with guard bits, will need to round when store accumulator into memory
- 3 DSP standard options
- Truncation: chop results
  => biases results up
- Round to nearest:
  < 1/2 round down, >= 1/2 round up (more positive)
  => smaller bais
- Convergent:
  < 1/2 round down, > 1/2 round up (more positive),
  = 1/2 round to make lsb a zero (+1 if 1, +0 if 0)
  => no bais
  IEEE 754 calls this round to nearest even

DSP Memory

- FIR Tap implies multiple memory accesses
- DSPs want multiple data ports
- Some DSPs have ad hoc techniques to reduce memory bandwidth demand
  - Instruction repeat buffer: do 1 instruction 256 times
  - Often disables interrupts, thereby increasing interrupt response time
- Some recent DSPs have instruction caches
  - Even then may allow programmer to "lock in" instructions into cache
  - Option to turn cache into fast program memory
- No DSPs have data caches
- May have multiple data memories

DSP Addressing

- Have standard addressing modes: immediate, displacement, register indirect
- Want to keep MAC datapth busy
- Assumption: any extra instructions imply clock cycles of overhead in inner loop
  => complex addressing is good
  => don't use datapath to calculate fancy address
- Autoincrement/Autodecrement register indirect
  - lw r1,0(r2)+ => r1 <- M[r2]; r2<-r2+1
  - Option to do it before addressing, positive or negative

DSP Addressing: Buffers

- DSPs dealing with continuous I/O
- Often interact with an I/O buffer (delay lines)
- To save memory, buffer often organized as circular buffer
- What can do to avoid overhead of address checking instructions for circular buffer?
- Option 1: Keep start register and end register per address register for use with autoincrement addressing, reset to start when reach end of buffer
- Option 2: Keep a buffer length register, assuming buffers starts on aligned address, reset to start when reach end
- Every DSP has "modulo" or "circular" addressing

DSP Addressing: FFT

- FFTs start or end with data in wierd bufferfly order

0 (000)	=>	0 (000)
1 (001)	=>	4 (100)
2 (010)	=>	2 (010)
3 (011)	=>	6 (110)
4 (100)	=>	1 (001)
5 (101)	=>	5 (101)
6 (110)	=>	3 (011)
7 (111)	=>	7 (111)

- What can do to avoid overhead of address checking instructions for FFT?
  - Have an optional "bit reverse" address addressing mode for use with autoincrement addressing
- Many DSPs have "bit reverse" addressing for radix-2 FFT

Addressing mode usage for DSP TI TMS320C54x for 54 library routines (static)

Addressing Mode	Percent	Running
Immediate	30.0%	30%
Autoincrement, post increment (incr. register after use contents as address)	18.8%	49%
Register indirect	17.4%	66%
Direct	12.0%	78%
Displacement	10.8%	89%
Autodecrement, post decrement (decr. register after use contents as address)	6.1%	95%
Autoincrement, post increment by contents of AR0, with circular addressing	2.2%	97%
Autoincrement, post increment by contents of AR0	1.5%	99%

- MIPS modes = 70%; autoinc,dec 25%; circular = 2.2%, bit reverse = 0.0%

DSP Instructions

- May specify multiple operations in a single instruction
- Must support Multiply-Accumulate (MAC)
- Need parallel move support
- Usually have special loop support to reduce branch overhead
  - Loop an instruction or sequence
  - 0 value in reigster usually means loop maximum number of times
  - Must be sure if calculate loop count that 0 does not mean 0
- May have saturating shift left arithmetic
- May have conditional execution to reduce branches

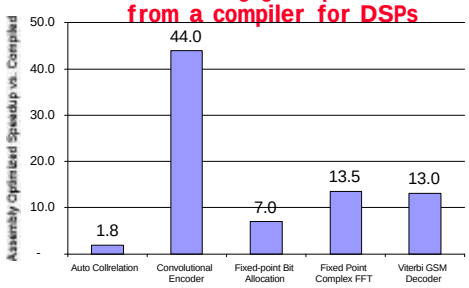
DSP vs. General Purpose MPU

- DSPs are like embedded MPUs, very concerned about energy and cost.
  - So concerned about cost is that they might even us a 4.0 micron (not 0.40) to try to shrink the the wafer costs by using fab line with no overhead costs.
- DSPs that fail are often claimed to be good for something other than the highest volume application, but that's just designers fooling themselves.
- Very recently convention wisdom has changed so that you try to do everything you can digitally at low voltage so as to save energy.
  - 1995 people thought doing everything in analog reduced power, but advances in lower power digital design flipped that wisdowm

DSP vs. General Purpose MPU

- The "MIPS/MFLOPS" of DSPs is speed of Multiply-Accumulate (MAC).
  - DSP are judged by whether they can keep the multipliers busy 100% of the time.
- The "SPEC" of DSPs is 4 algorithms:
  - Inifinite Impule Response (IIR) filters
  - Finite Impule Response (FIR) filters
  - FFT, and
  - convolvers
- In DSPs, algorithms are king!
  - Binary compatability not an issue
- Software is not (yet) king in DSPs.
  - People still write in assembly language for a product to minimize the die area for ROM in the DSP chip.

Pitfall: Getting good performance from a compiler for DSPs



Telecom EEMBC Kernels for Texas Instructments TMS320C6203-300MHz DSP

- ~10X in performance assembly vs. compiled for TI DSP (May 2000 EEMBC results)

Summary: How are DSPs different?

- Essentially infinite streams of data which need to be processed in real time
- Relatively small programs and data storage requirements
- Intensive arithmetic processing with low amount of control and branching (in the critical loops)
- High amount of I/O with analog interface
- Loosely coupled multiprocessor operation is popular: 1 program/CPU vs. SPMD?

Summary: How are DSPs different?

- Single cycle multiply accumulate (multiple busses and array multipliers)
- Complex instructions for standard DSP functions (IIR and FIR filters, convolvers)
- Specialized memory addressing
  - Modular arithmetic for circular buffers (delay lines)
  - Bit reversal (FFT)
- Zero overhead loops and repeat instructions
- I/O support – Serial and parallel ports

Summary: Unique Features in DSP architectures

- Continuous I/O stream, real time requirements
- Multiple memory accesses
- Autoinc/autodec addressing
- Datapath
  - Multiply width
  - Wide accumulator
  - Guard bits/shiting rounding
  - Saturation
- Weird things
  - Circular addressing
  - Reverse addressing
- Special instructions
  - shift left and saturate (arithmetic left-shift)

DSP Summary 2

- DSP processor performance has increased by a factor of about 150x over the past 15 years (~40%/year)
- Processor architectures for DSP will be increasingly specialized for applications, especially communication applications
- General-purpose processors will become viable for many DSP applications
- Users of processors for DSP will have an expanding array of choices

For More DSP Information

- http://www.bdti.com
  Collection of BDTI's papers on DSP processors, tools, and benchmarking.
- http://www.eg3.com/dsp
  Links to other good DSP sites.
- *Microprocessor Report*
  For info on newer DSP processors.
- *DSP Processor Fundamentals*,
  Textbook on DSP Processors, BDTI
- *IEEE Spectrum*, July, 1996
  Article on DSP Benchmarks
- *Embedded Systems Prog.*, October, 1996
  Article on Choosing a DSP Processor