McGill

ECSE 421 Lecture 9: Processing Elements

ESD Chapter 3

Last Time

- Sensors
- Discretization
  - Time: sample-and-hold
  - Values: ADC

Where Are We?

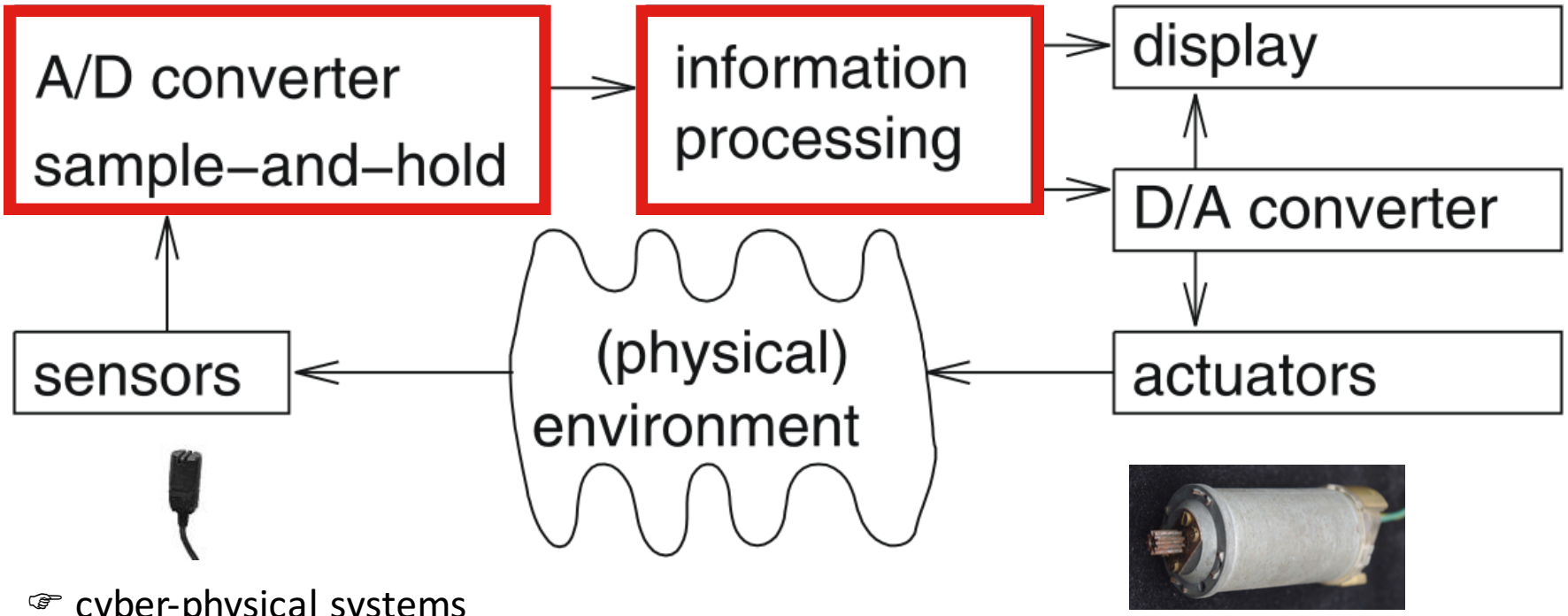
W	D	Date		Topic	ESD	PES	Out	In	Notes
1	T	12-Jan-2016	L01	Introduction to Embedded System Design	1.1-1.4				
	R	14-Jan-2016		Introduction to Embedded System Design	1.1-1.4				
2	T	19-Jan-2016	L02	Specifying Requirements / MoCs / MSC	2.1-2.3				
	R	21-Jan-2016	L03	CFSMs	2.4				
3	T	26-Jan-2016	L04	Data Flow Modeling	2.5	3.1-5,7	LA1		
	R	28-Jan-2016	L05	Petri Nets	2.6				Thru Slide 21
4	T	2-Feb-2016	L06	Discrete Event Models	2.7	4			G: Zaid Al-bayati
	R	4-Feb-2016	L07	DES / Von Neumann Model of Computation	2.8-2.10	5	LA2	LA1	
5	T	9-Feb-2016	L08	Sensors	3.1-3.2	7.3,12.1-6			
	R	11-Feb-2016	L09	Processing Elements	3.3	12.6-12			
6	T	16-Feb-2016	L10	More Processing Elements / FPGAs			LA3	LA2	
	R	18-Feb-2016	L11	Memories, Communication, Output	3.4-3.6				
7	T	23-Feb-2016	L12	Embedded Operating Systems	4.1				
	R	25-Feb-2016		***Midterm exam: in-class, closed book***			P	LA3	Chapters 1-3
	T	1-Mar-2016		***No class***					Winter break
	R	3-Mar-2016		***No class***					Winter break
8	T	8-Mar-2016	L13	Middleware	4.4-4.5				
	R	10-Mar-2016	L14	Performance Evaluation	5.1-5.2				
9	T	15-Mar-2016	L15	More Evaluation and Validation	5.3-5.8				
	R	17-Mar-2016	L16	Introduction to Scheduling	6.1-6.2.2				
10	T	22-Mar-2016	L17	Scheduling Aperiodic Tasks	6.2.3-6.2.4				
	R	24-Mar-2016	L18	Scheduling Periodic Tasks	6.2.5-6.2.6				

Today

- Embedded Information Processing

Embedded System Hardware

- Embedded system hardware is frequently used in a loop ("hardware in a loop"):



☞ cyber-physical systems

Processing Units

- Need for efficiency (power + energy):

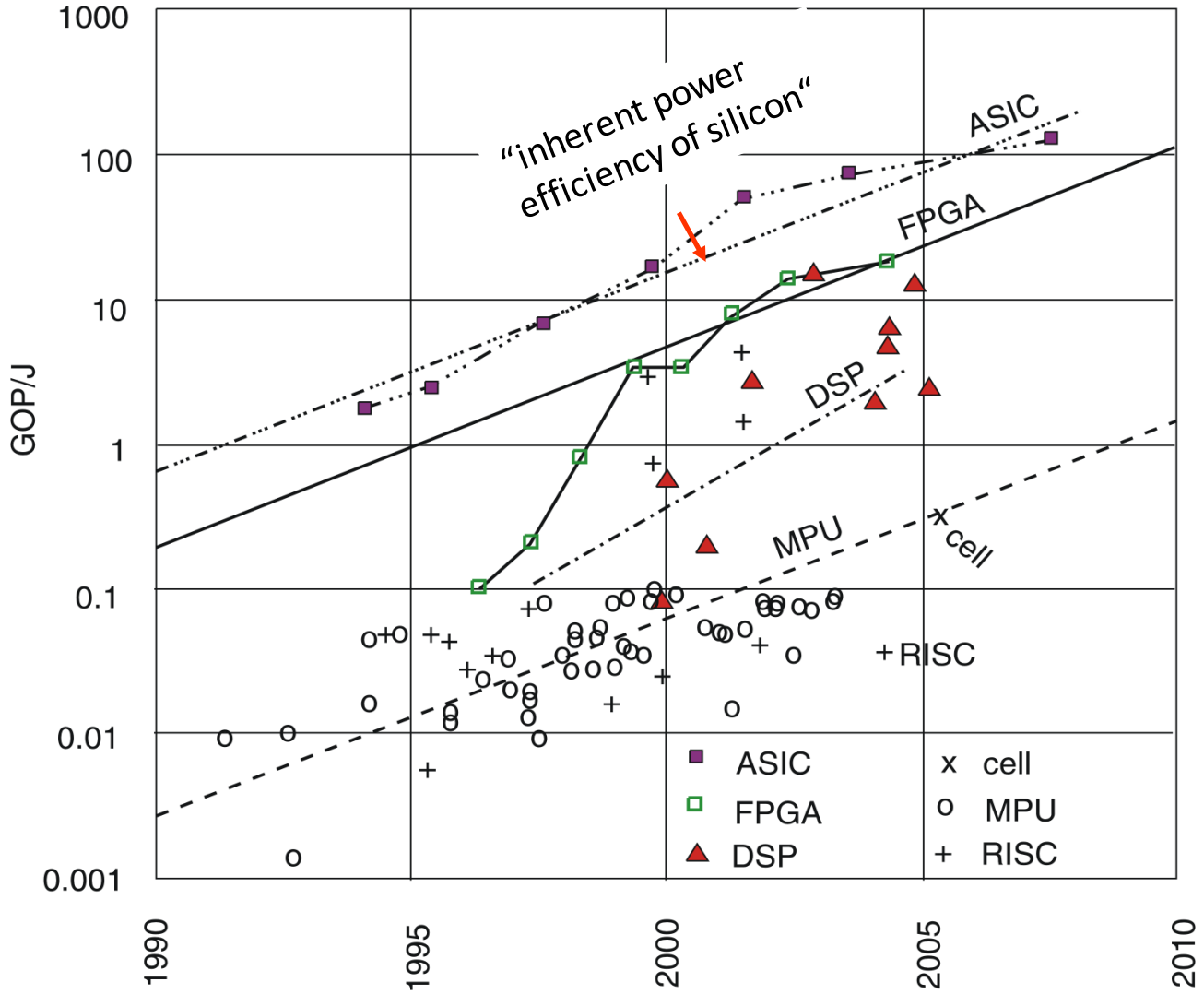
  "Power is considered as the most important constraint in embedded systems"
  [in: L. Eggermont (ed): Embedded Systems Roadmap 2002, STW]

Why worry about energy and power?

Energy consumption by IT is the key concern of green computing initiatives (embedded computing is leading the way)

http://www.esa.int/images/earth,4.jpg

Importance of Energy Efficiency

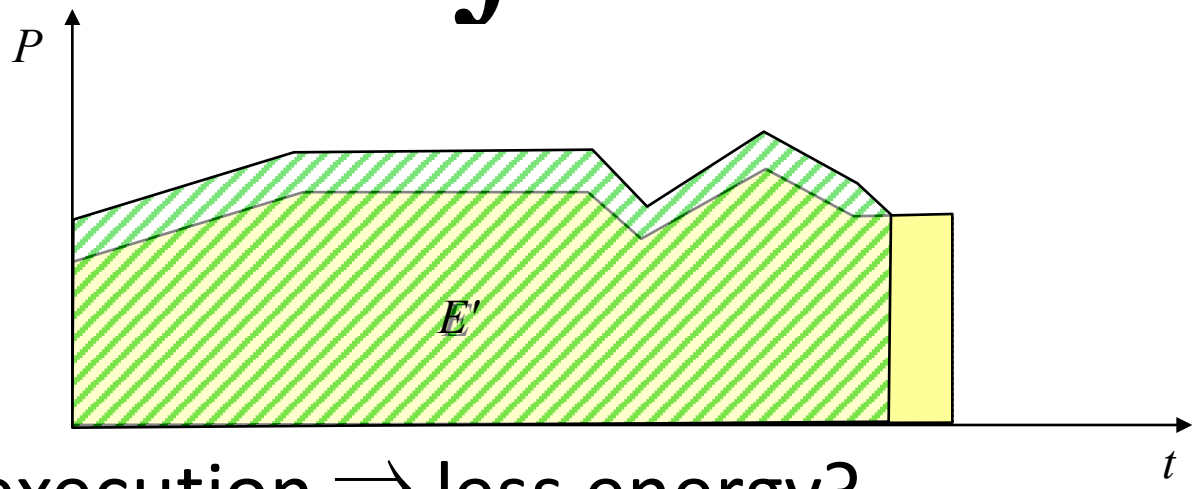


© Hugo De Man, IMEC, Philips, 2007

Efficient software design needed, otherwise, the price for software flexibility cannot be paid.

Power and Energy Are Related

$$E = \int P \, dt$$



- Faster execution ⇒ less energy?
  - Sometimes
  - But not if power is increased significantly to enable faster execution (s.t. $E' > E$)

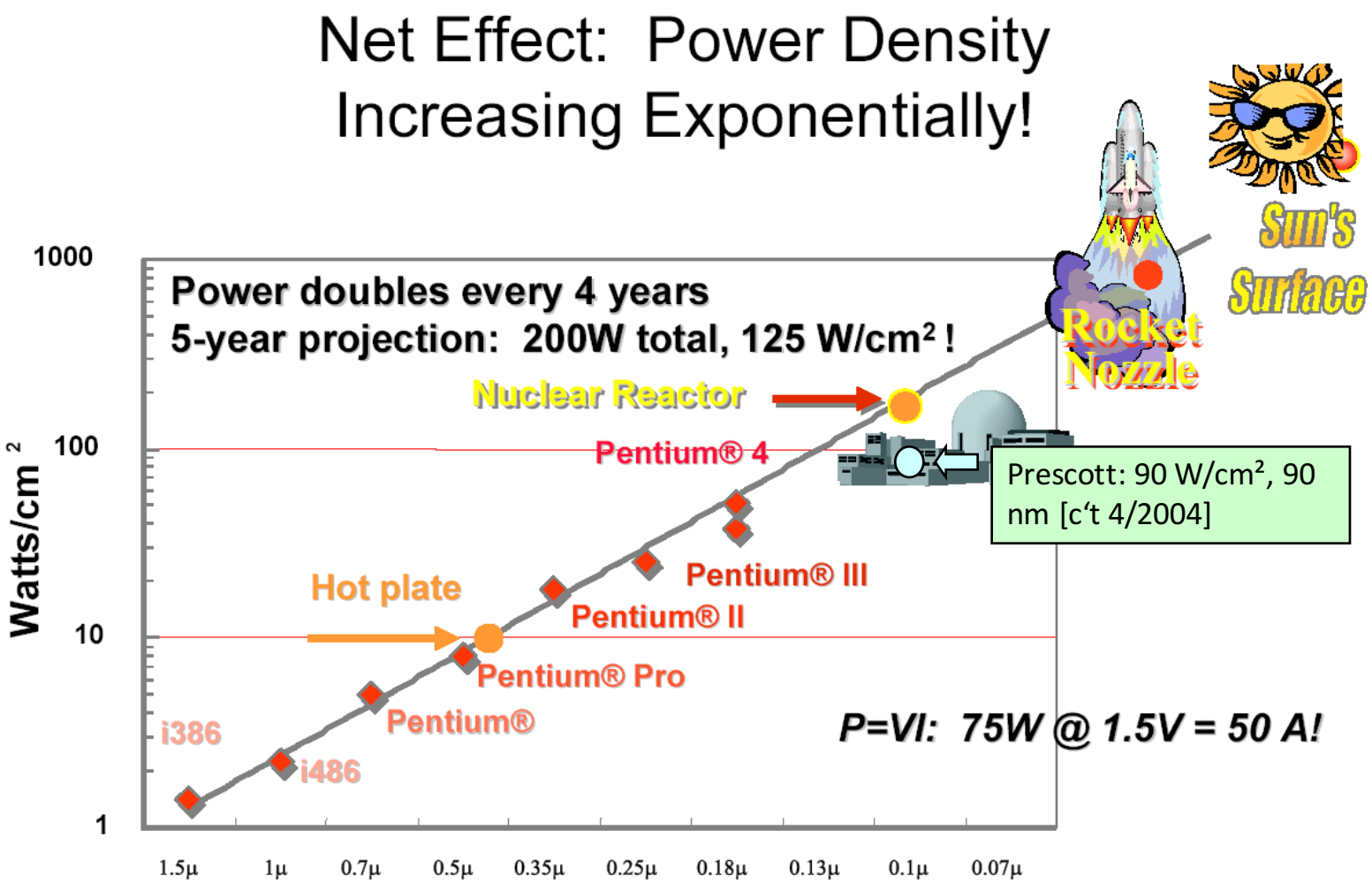
Low Power vs. Low Energy (1)

- Reducing **power consumption** important for
  - the design of the power supply
  - the design of voltage regulators
  - the dimensioning of interconnect
  - short term cooling

Low Power vs. Low Energy (2)

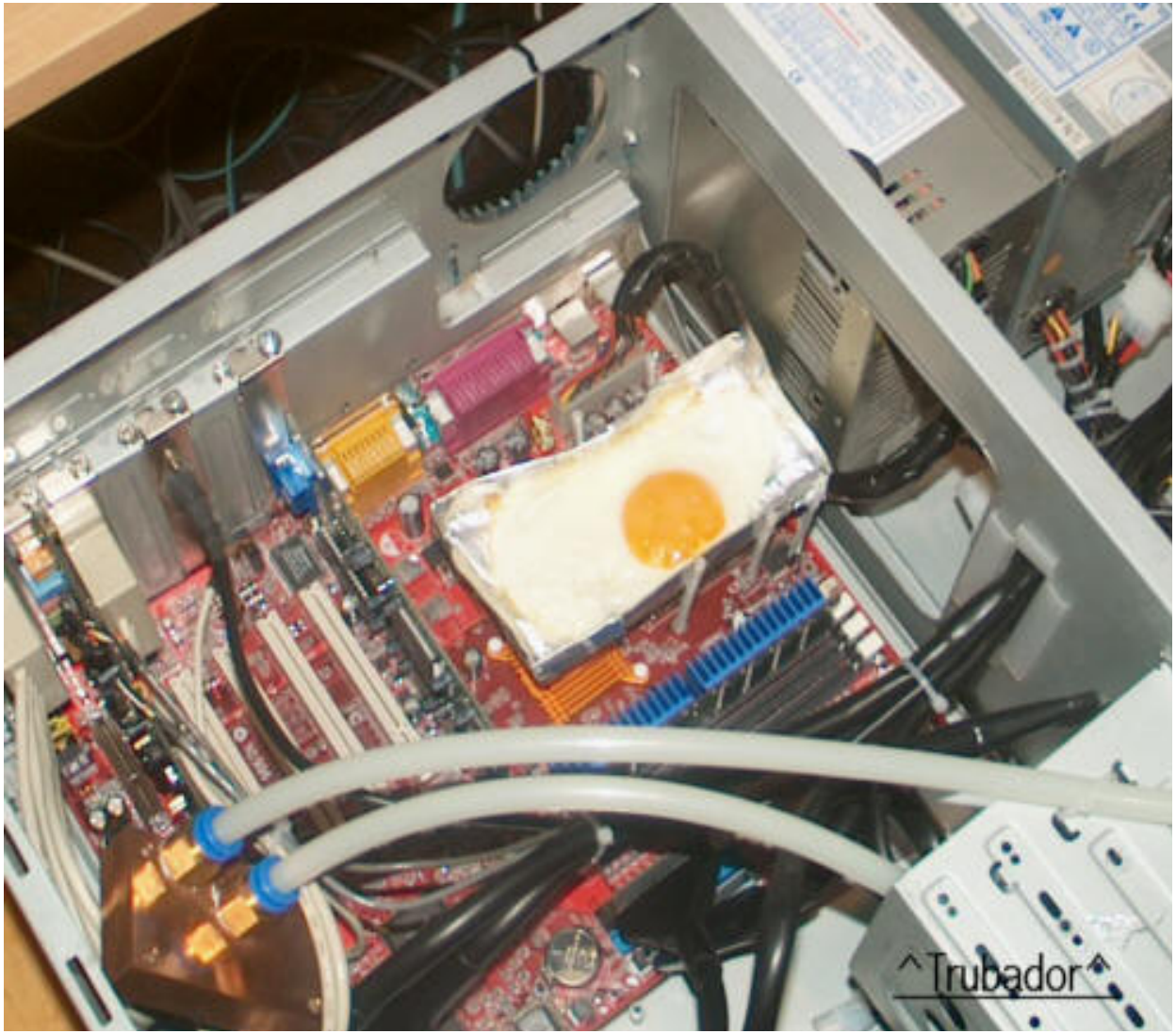
- Reducing **energy consumption** important due to
  - restricted availability of energy (mobile systems)
    - limited battery capacities (only slowly improving)
    - very high costs of energy (solar panels, in space)
  - cooling
    - high costs
    - limited space
    - low noise requirements
  - dependability
  - long lifetimes, low temperatures

Technology Scaling and Power Denisty



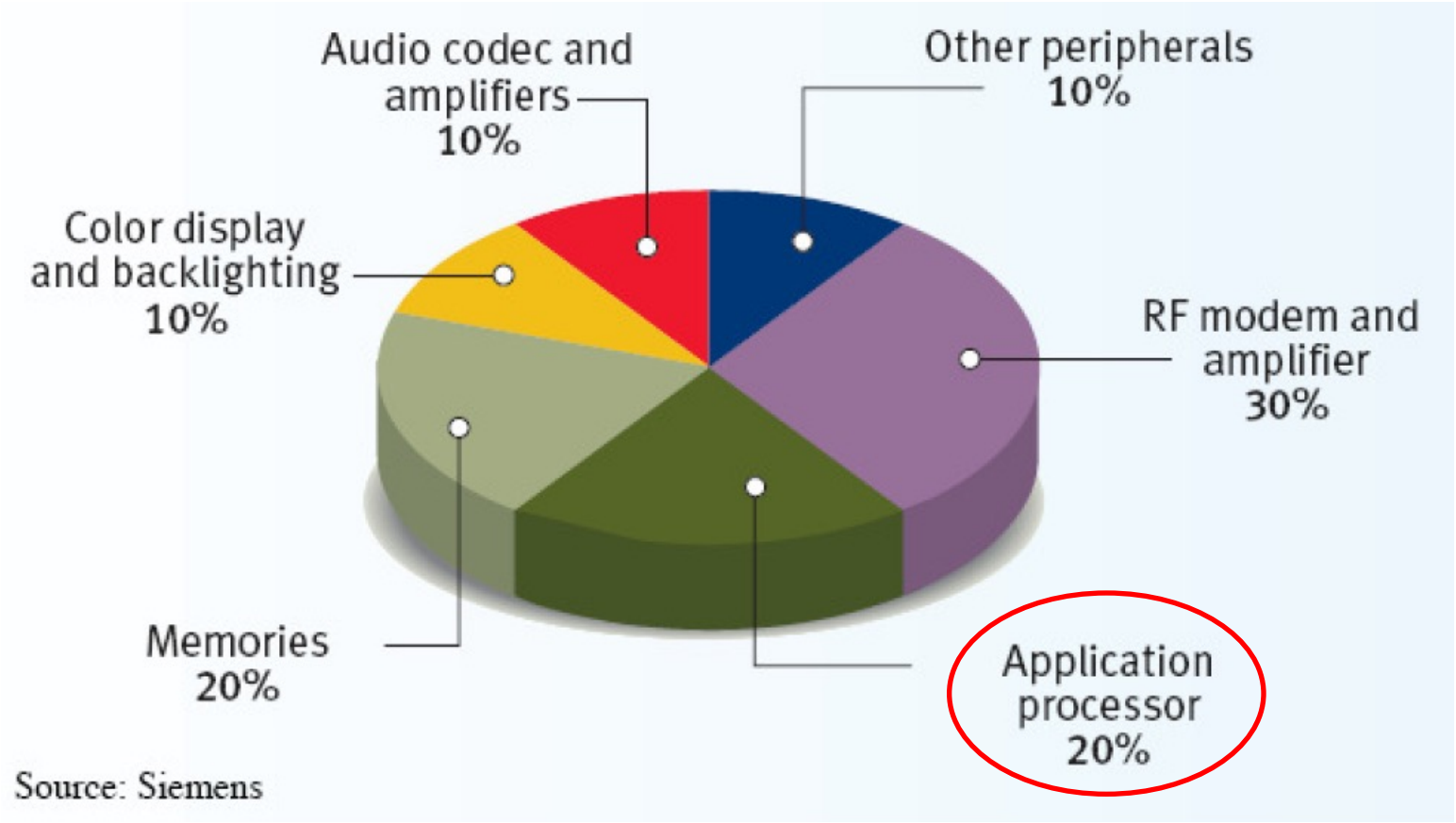
* "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies" – Fred Pollack, Intel Corp. Micro32 conference key note - 1999. Courtesy Avi Mendelson, Intel.

Hotter than a hot plate? Why not …



http://www.phys.ncku.edu.tw/~htsu/humor/fry_egg.html

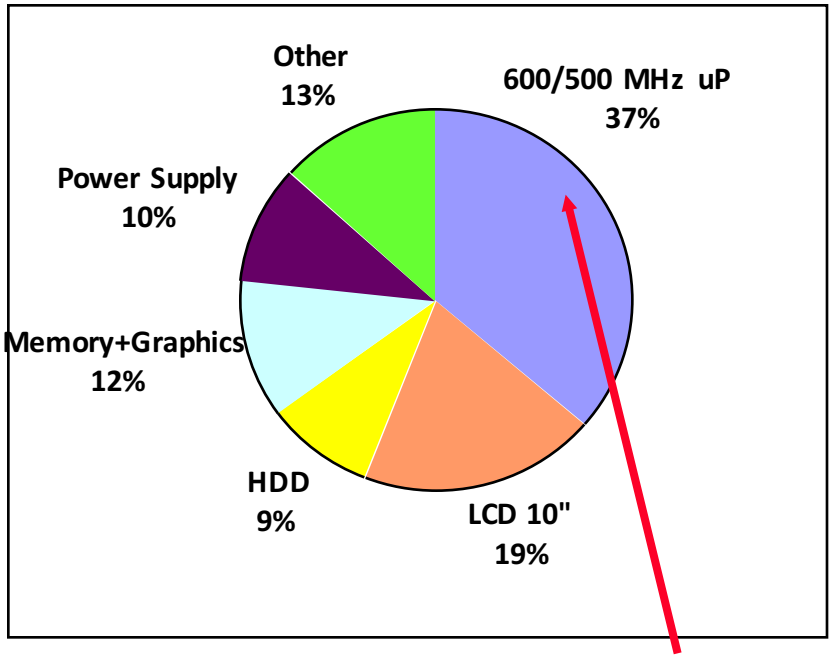
Energy Consumption in Mobile Devices



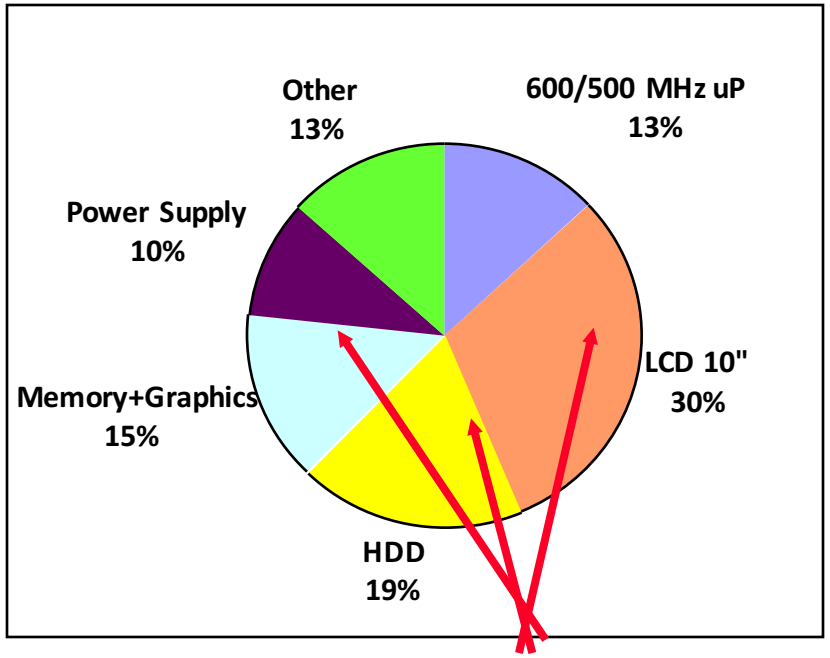
[O. Vargas (Infineon Technologies): Minimum power consumption in mobile-phone memory subsystems; Pennwell Portable Design - September 2005;] Thanks to Thorsten Koch (Nokia/ TU Dortmund) for providing this source.

CPU vs. System Power

Mobile PC (notebook)
Thermal Design (TDP) System Power



Mobile PC (notebook)
Average System Power



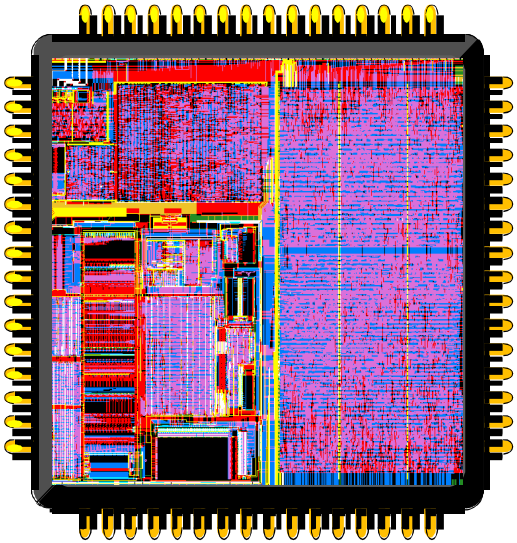
Note: Based on Actual Measurements

CPU Dominates Thermal Design Power

Multiple Platform Components Comprise Average Power

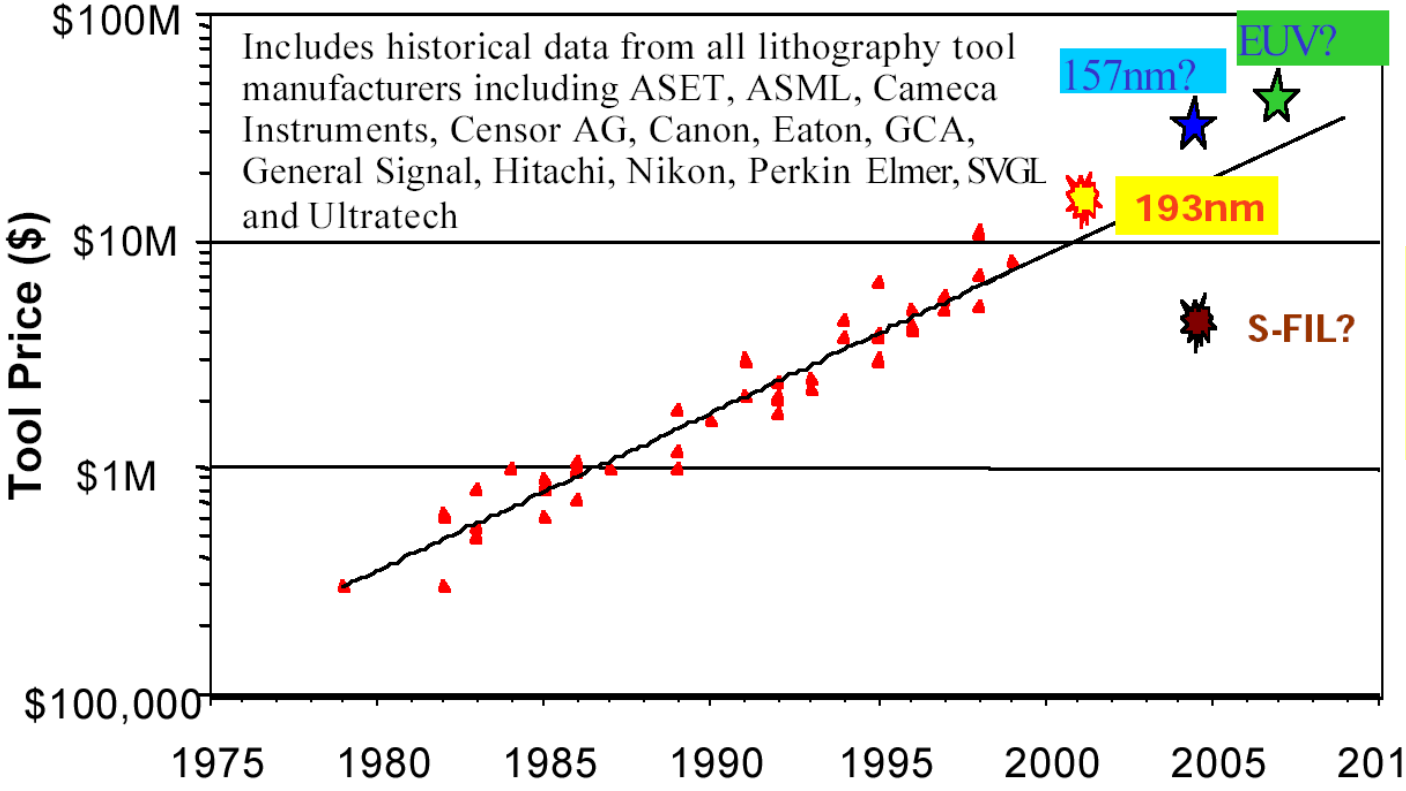
[Courtesy: N. Dutt; Source: V. Tiwari]

Application Specific Integrated Circuits



- ASICS, or full-custom designs
- Suffer from
  - long design times
  - lack of flexibility (changing standards)
  - high costs (e.g. M$ mask costs)
- Custom-designed circuits are useful only when
  - the highest performance possible is needed,
  - the greatest energy efficiency is the goal, and
  - large numbers can be sold

Trends in NRE (e.g., Masks)



⇒Trend toward SW and FPGA implementations

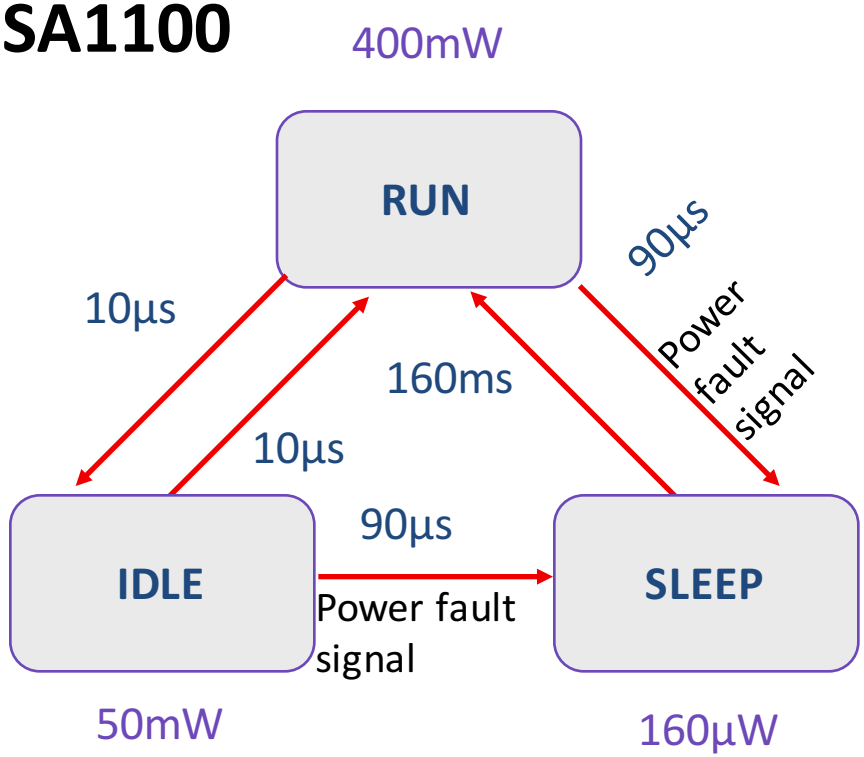
HW synthesis not covered in this course.

[http://www.molecularimprints.com/Technology/tech_articles/MII_COO_NIST_2001.PDF9]

Key Requirements for Processors

1. Energy/ power-efficiency

- Dynamic power management (DPM)
  - **Example: STRONGARM SA1100**
- RUN: operational
- IDLE: stop the CPU when not in use; monitor interrupts
- SLEEP: Shutdown of on-chip activity



Dynamic Voltage Scaling (DVS)

CMOS Power Consumption (ignoring leakage):

$P = \alpha\ C_L\ V_{dd}^2\ f$ with

α : switching activity

C_L : load capacitance

V_{dd} : supply voltage

f : clock frequency

Delay for CMOS circuits:

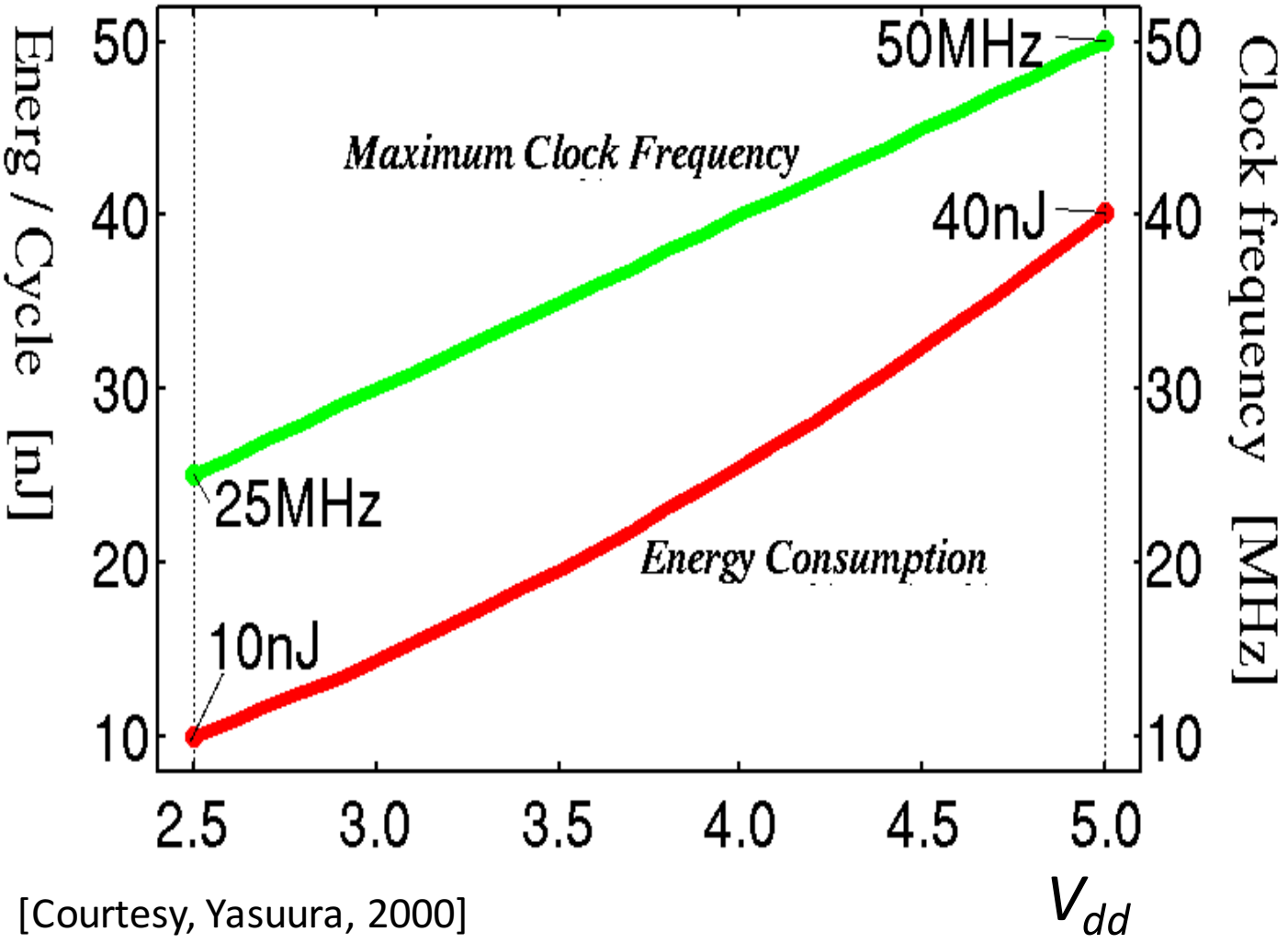
$$\tau = k\, C_L \frac{V_{dd}}{(V_{dd} - V_t)^2}$$ with

V_t : threshhold voltage

$(V_t$ < than $V_{dd})$

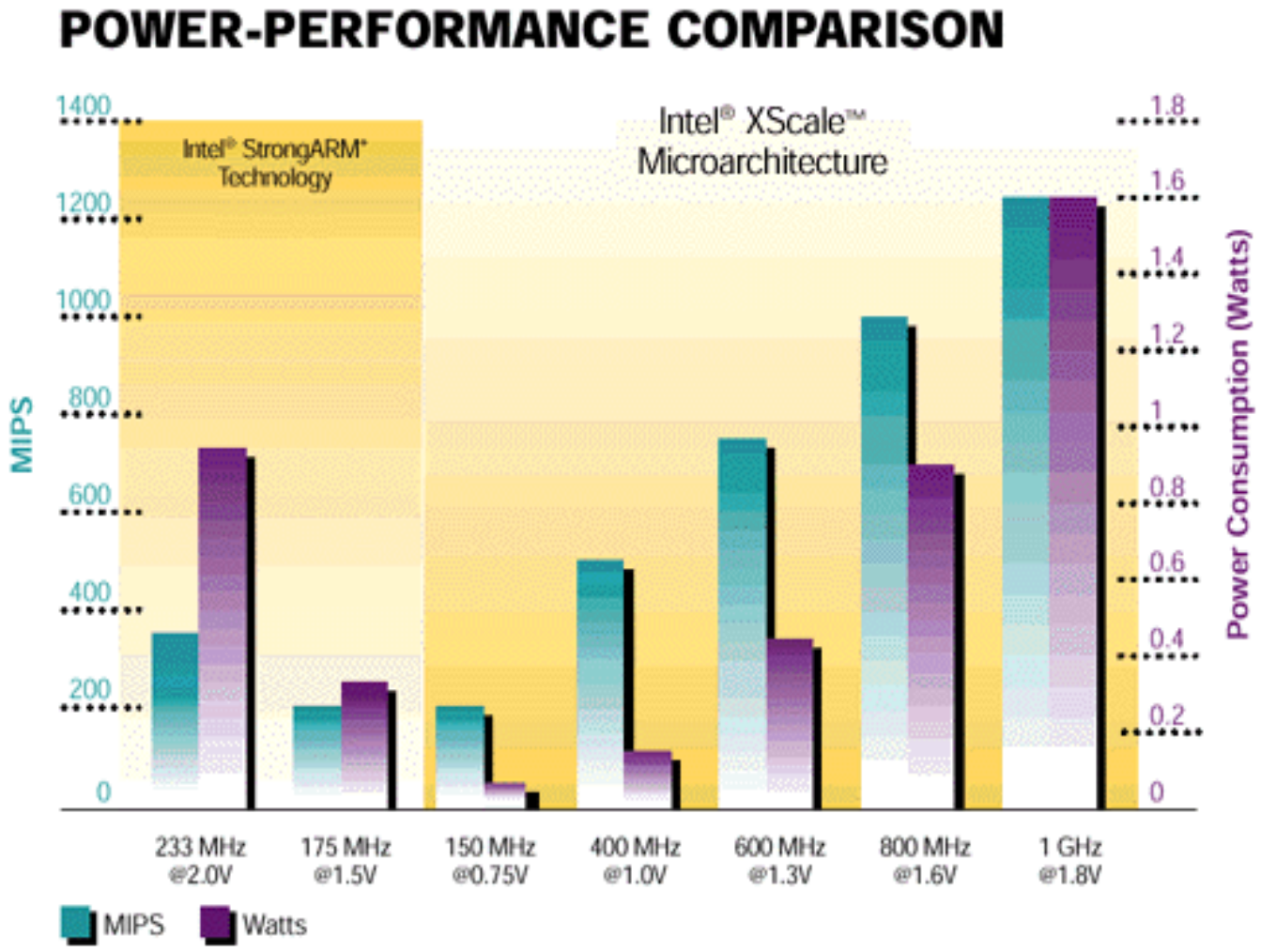
$\Rightarrow$ Decreasing V_{dd} reduces P *quadratically*, and increases latency *linearly*

Voltage Scaling: Example



[Courtesy, Yasuura, 2000]

DVFS Example: Intel XScale



OS should schedule how the energy budget is distributed

[Source: Intel]

Low Voltage, Parallel Operation

Basic equations

Power: $P \sim V_{DD}^2$,

Maximum clock frequency: $f \sim V_{DD}$,

Energy to run a program: $E = P \times t$, with: t = runtime (fixed)

Time to run a program: $t \sim 1/f$

Changes due to parallel processing, with α operations per clock

Clock frequency reduced to: $f' = f / \alpha$,

Voltage can be reduced to: $V_{DD}' = V_{DD} / \alpha$,

Power for parallel processing: $P^\circ = P / \alpha^2$ per operation,

Power for α operations per clock: $P' = \alpha \times P^\circ = P / \alpha$,

Time to run a program is still: $t' = t$,

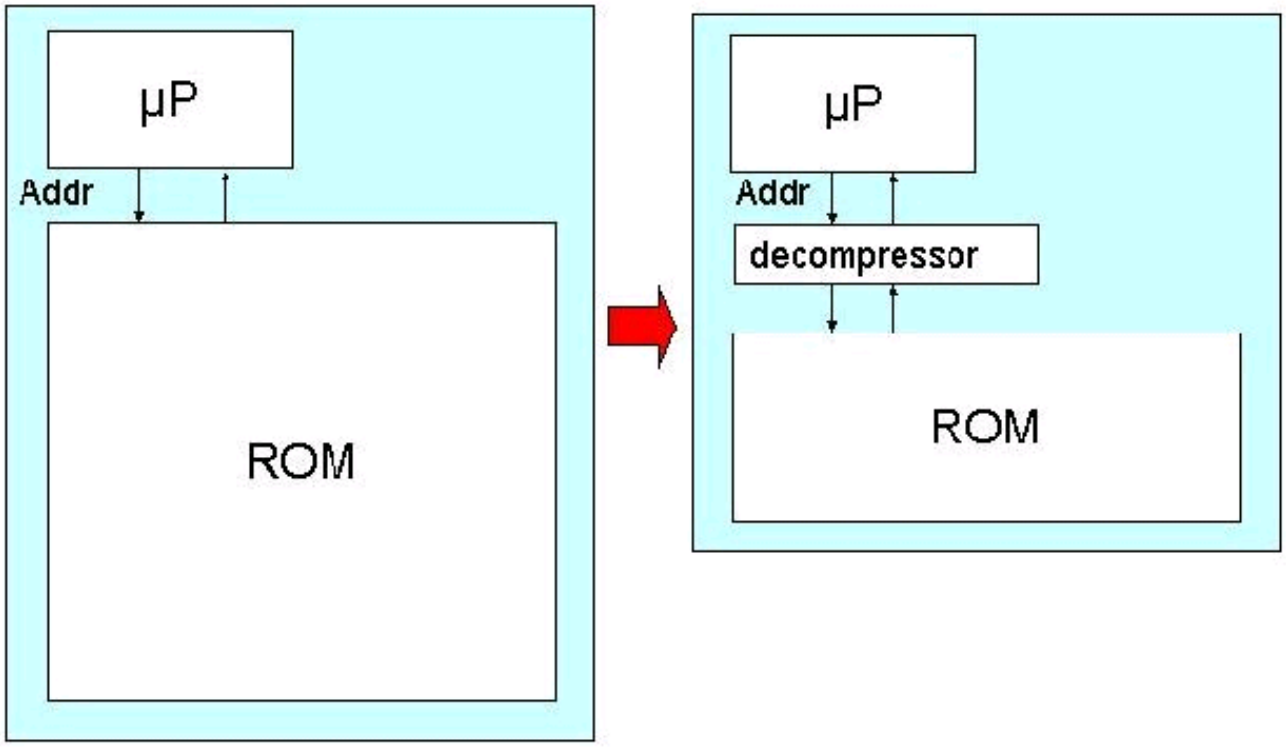
Energy required to run program: $E' = P' \times t = E / \alpha$

$\Rightarrow$ Argument in favour of voltage scaling, VLIW processors, and multi-core systems

Rough approximations!

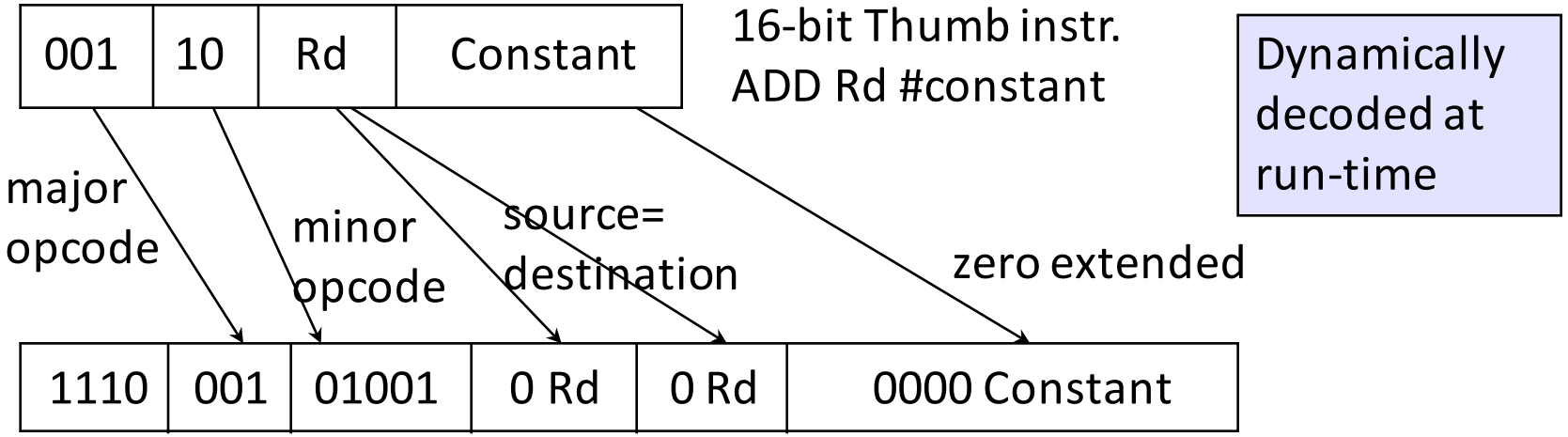
Key Requirements for Processors

2. Code-size efficiency
   - CISC machines: complex instructions reduce code size
   - RISC machines: improve performance at the expense of code size
   - Alternative solution? RISC + Compression



Code-size Efficiency with Alternative ISA

- Reduced-width instruction set
  - *E.g.* ARM Thumb



- Reduction to 65-70% of original code size
- 130% of ARM performance with 8/16 bit memory
- 85% of ARM performance with 32-bit memory

[ARM, R. Gupta]

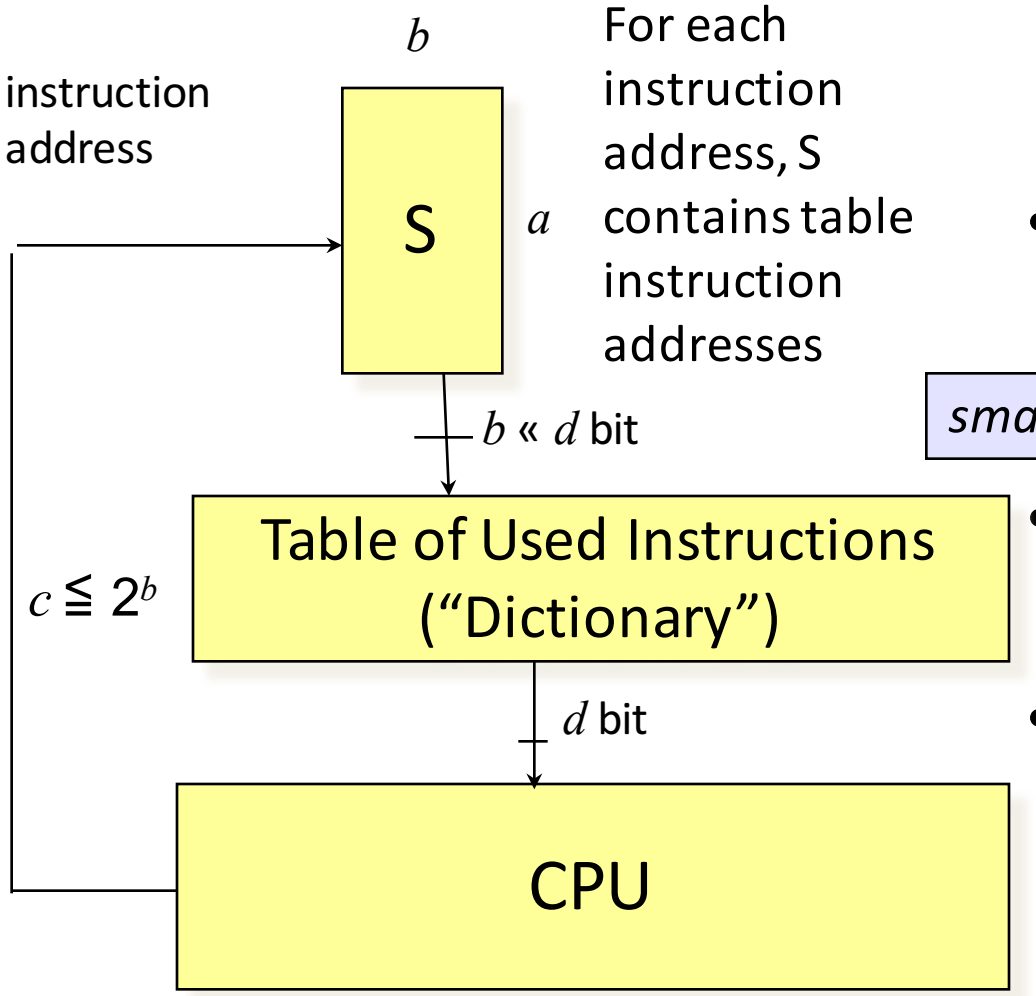
Same approach for LSI TinyRisc, …
Requires support by compiler, assembler, etc.

Code-size Efficiency with Dictionary

"Dictionary-based coding … use some kind of a dictionary that contains parts of the input sequence which frequently appear. The encoded sequence in turn contains references to the dictionary elements rather than containing these over and over."

[Á. Beszédes et al.: Survey of Code size Reduction Methods, Survey of Code-Size Reduction Methods, *ACM Computing Surveys*, Vol. 35, Sept. 2003]

Dictionary-based Compression



- Uncompressed storage
  - a d-bit-wide instructions $\Rightarrow$ axd bits
- Code compression
  - Each instruction pattern is stored once

small

- If $axb + cxd < axd$, code is compressed
- Nanoprogramming! (Motorola 68000)

Cache-based Compression

- Main idea: *decompress* cache lines upon fetch from memory
- Cache lines ↔ variable-sized blocks in memory
  - Line address tables (LATs) for translation of instruction addresses into memory addresses
- Tables may become large and have to be bypassed by a line address translation buffer

[A. Wolfe, A. Chanin, MICRO-92]

More Code Compaction References

- Popular code compaction library by Rik van de Wiel [http://www.extra.research.philips.com/ccb] has been moved to

  http://www-perso.iro.umontreal.ca/~latendre/codeCompression/codeCompression/node1.html

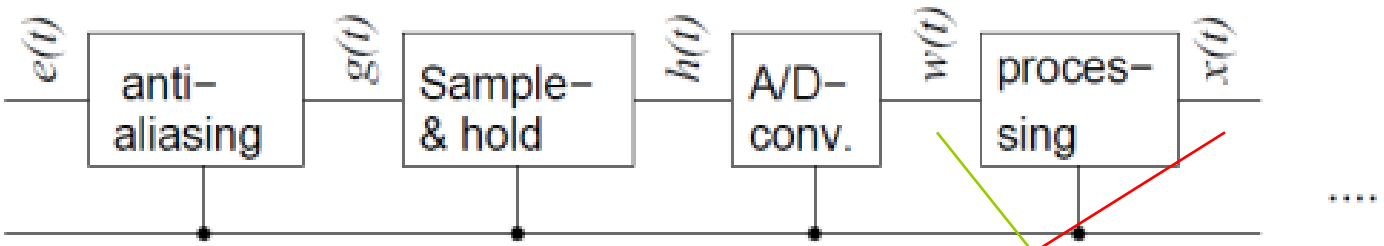
  http://www.iro.umontreal.ca/~latendre/compactBib/

  (153 entries as of 11/2004)

Key Requirements for Processors

3. Run-time efficiency
   – Domain-oriented architectures

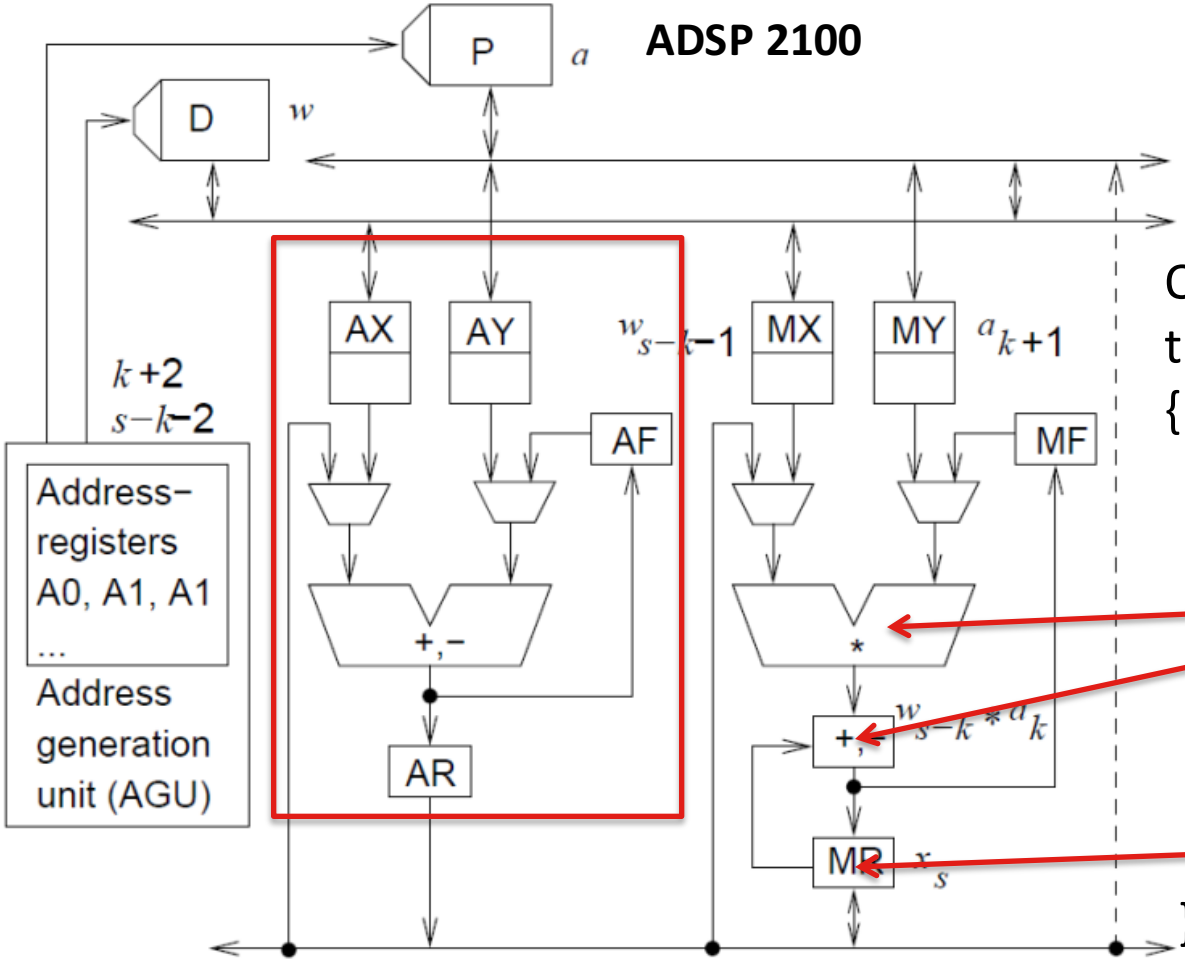
- Example: filtering in digital signal processing (DSP)



$$x_s = \sum_{k=0}^{n-1} w_{s-k} * a_k$$

Signal at $t=t_s$ (sampling points)

Filtering in DSP



$$x_s = \sum_{k=0}^{n-1} w_{s-k} * a_k$$

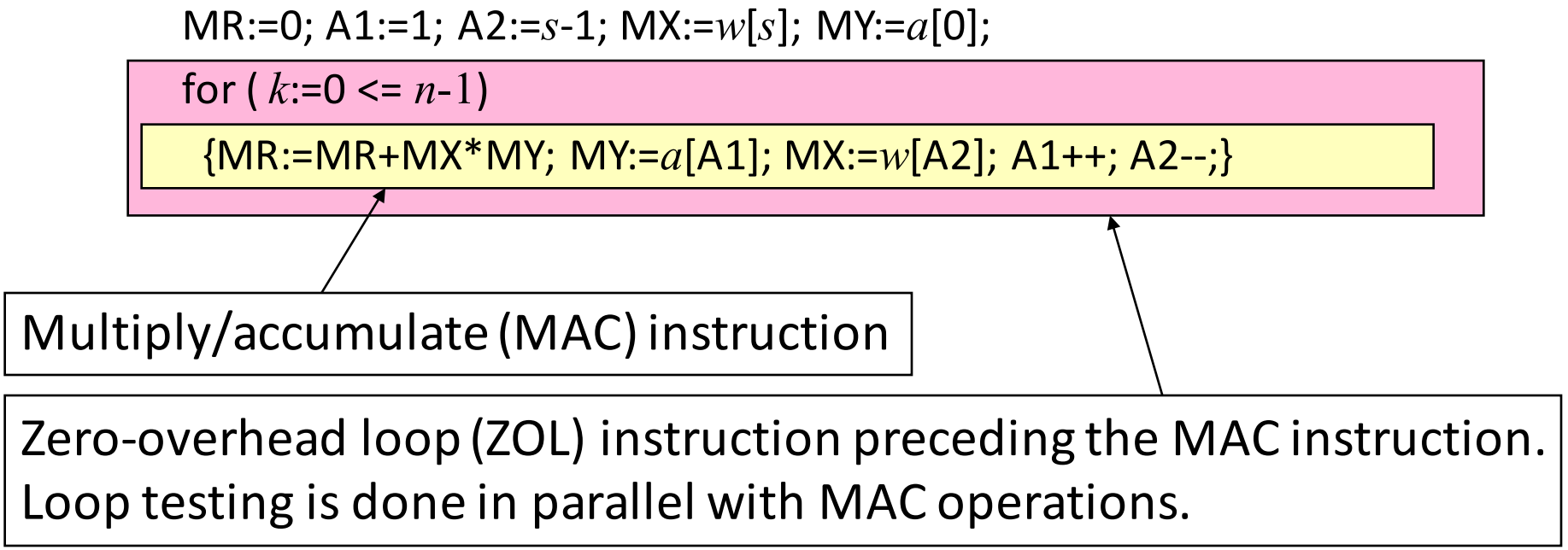
Outer loop over sampling times t_s:

```
{ MR:=0; A1:=1; A2:=s-1;
  MX:=w[s]; MY:=a[0];
  for (k=0; k <= (n−1); k++)
   { MR:=MR + MX * MY;
     MX:=w[A2]; MY:=a[A1];
     A1++; A2−−;
   }
  x[s]:=MR;
}
```

Maps nicely!

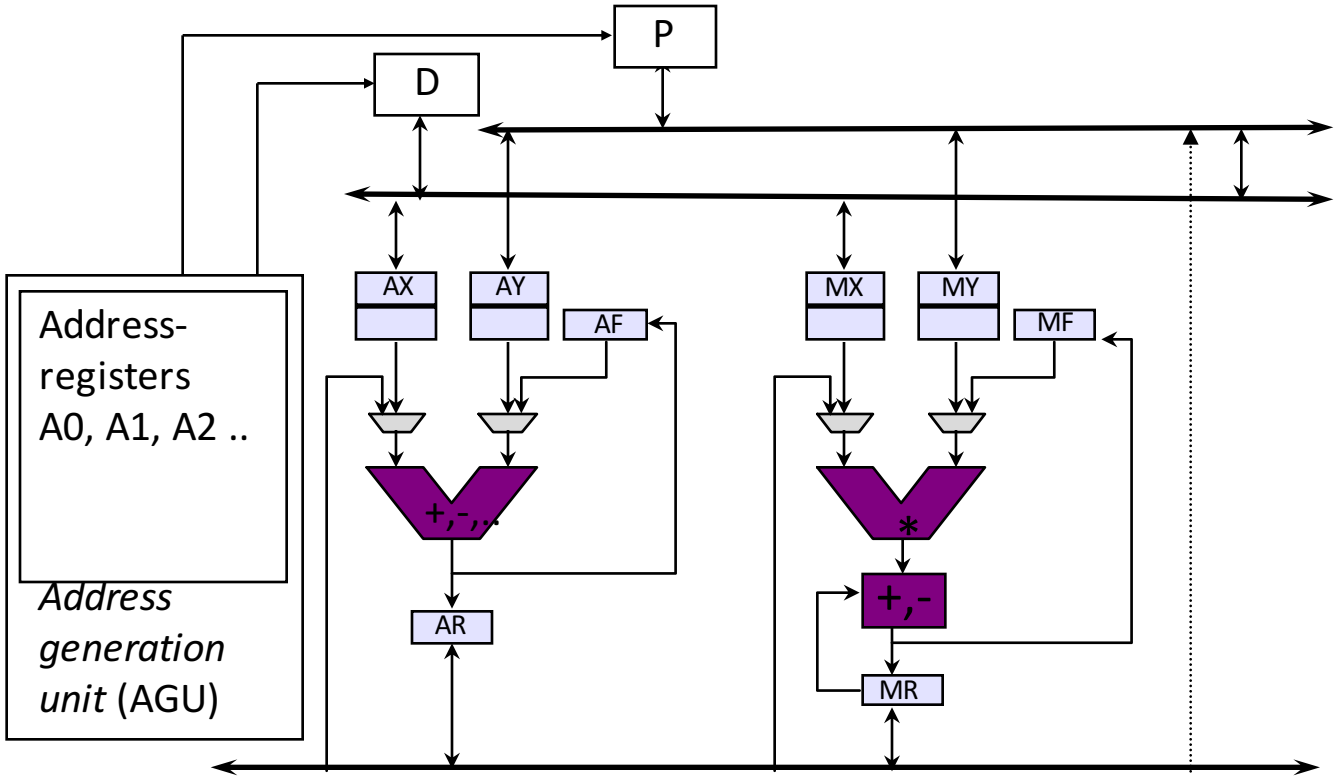
Common DSP Processor Optimizations

- Multiply/accumulate (MAC)
- Zero-overhead loop (ZOL)



Heterogeneous Registers

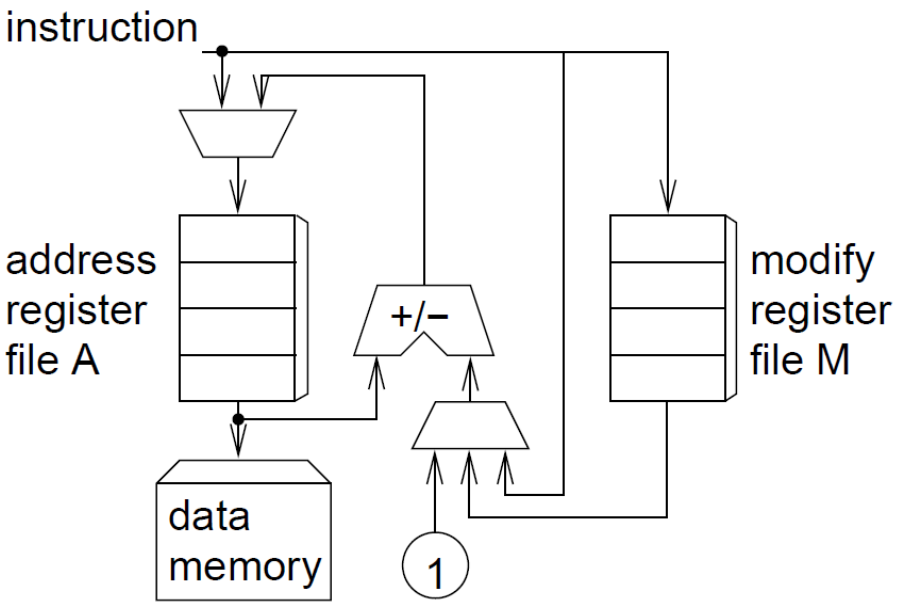
Example (ADSP 210x):



Different functionality for registers An, AX, AY, AF, MX, MY, MF, MR

Address Generation Units (AGUs)

Example (ADSP 210x):



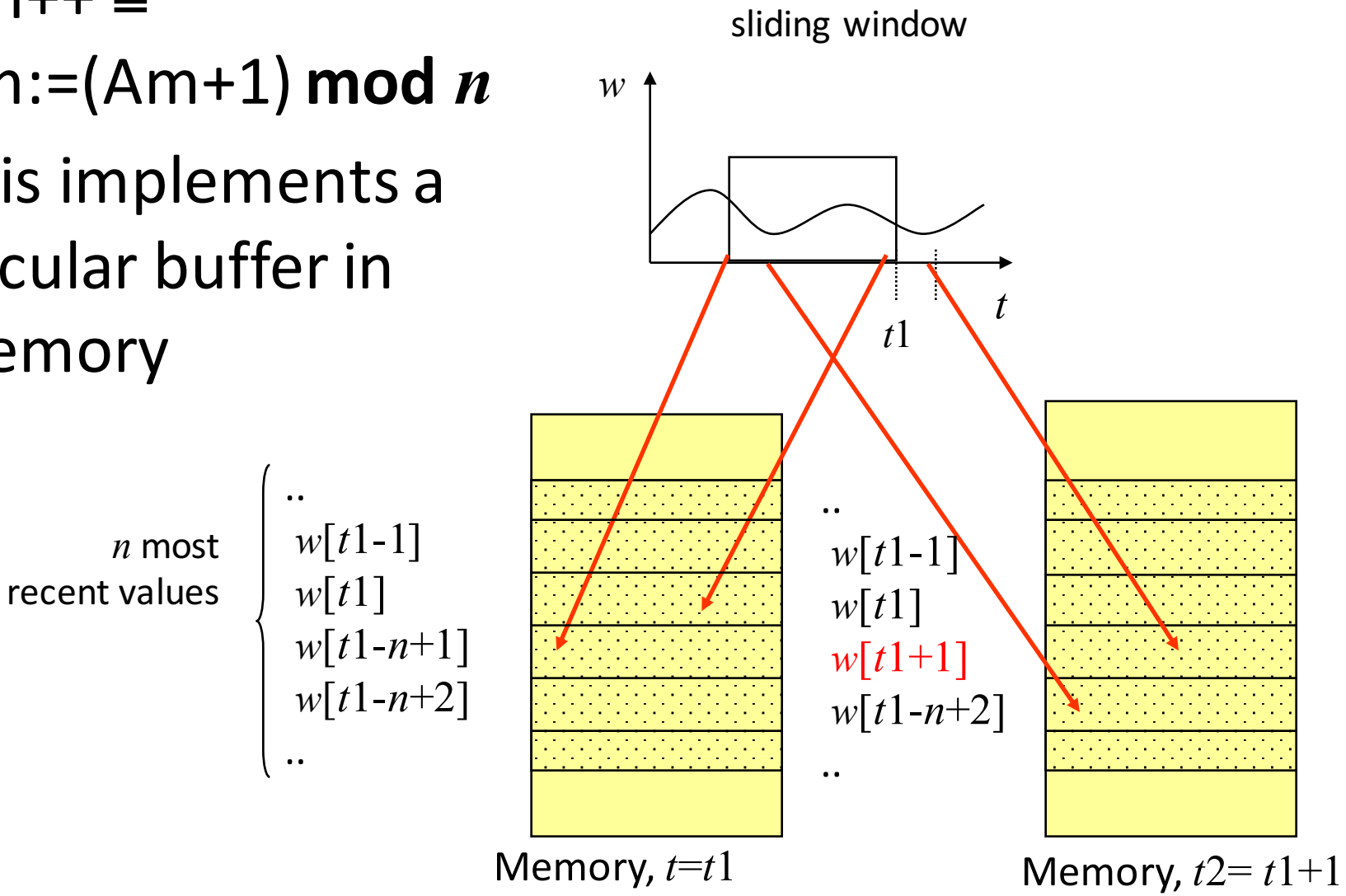
- Data memory is accessed with addresses in A
  - In parallel with operation in main data path (0 time)
- Address ALU: A := A ± 1 also takes 0 time
  - same for A := A ± M;
- A := <immediate in instruction> requires an extra instruction

⇒ Minimize use of load imm

⇒ Optimization in Chapter 7

Modulo Addressing

- Am++ ≡ Am:=(Am+1) **mod** ***n***
- This implements a circular buffer in memory



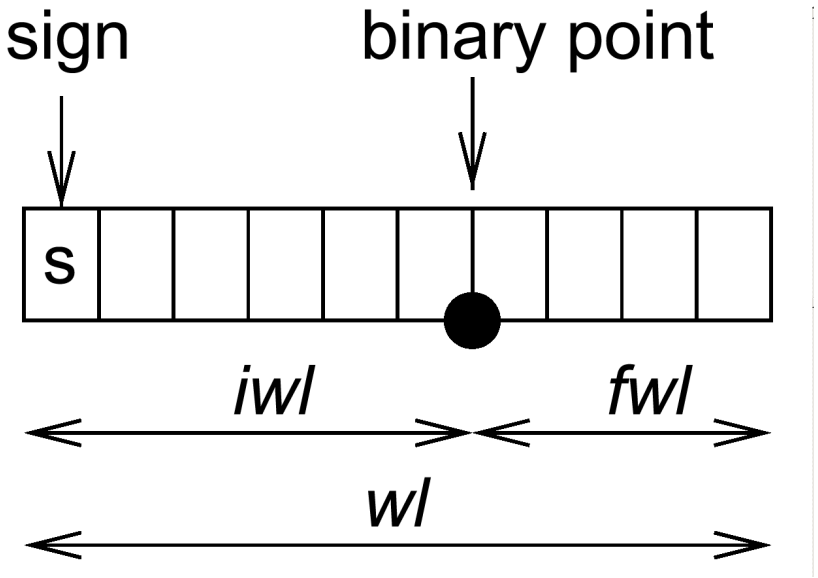
Saturating Arithmetic

- Returns the largest/smallest value when arithmetic over/underflows
- Examples:

a		0111	
b		+ 1001	
standard wrap around arithmetic		(1)0000	
saturating arithmetic		1111	
(a+b)/2:	correct	1000	
	wrap around arithmetic	0000	
	saturating arithmetic + shifted	0111	"Almost correct"

- Appropriate for DSP/multimedia applications:
  - If overflow results in an exception ⇒ results are available *late*
  - Precise values are often less important
  - Wrap around arithmetic would be worse

Fixed-point Arithmetic



Shifting required after multiplications and divisions in order to maintain binary point.

Properties of Fixed-point Arithmetic

- Automatic scaling important for multiplications
- Example:

  $x = 0.5 \times 0.125 + 0.25 \times 0.125 = 0.0625 + 0.03125 = 0.09375$

  – For *iwl*=1 and *fwl*=3, the least significant digits are automatically truncated: $x = 0.093$

  – Acts like a floating point system with numbers $\in$ (-1...1), with no stored exponent (bits used to increase precision)
- Appropriate for DSP/multimedia applications

  – Well-known value ranges
- More efficient than implementing a general FPU!

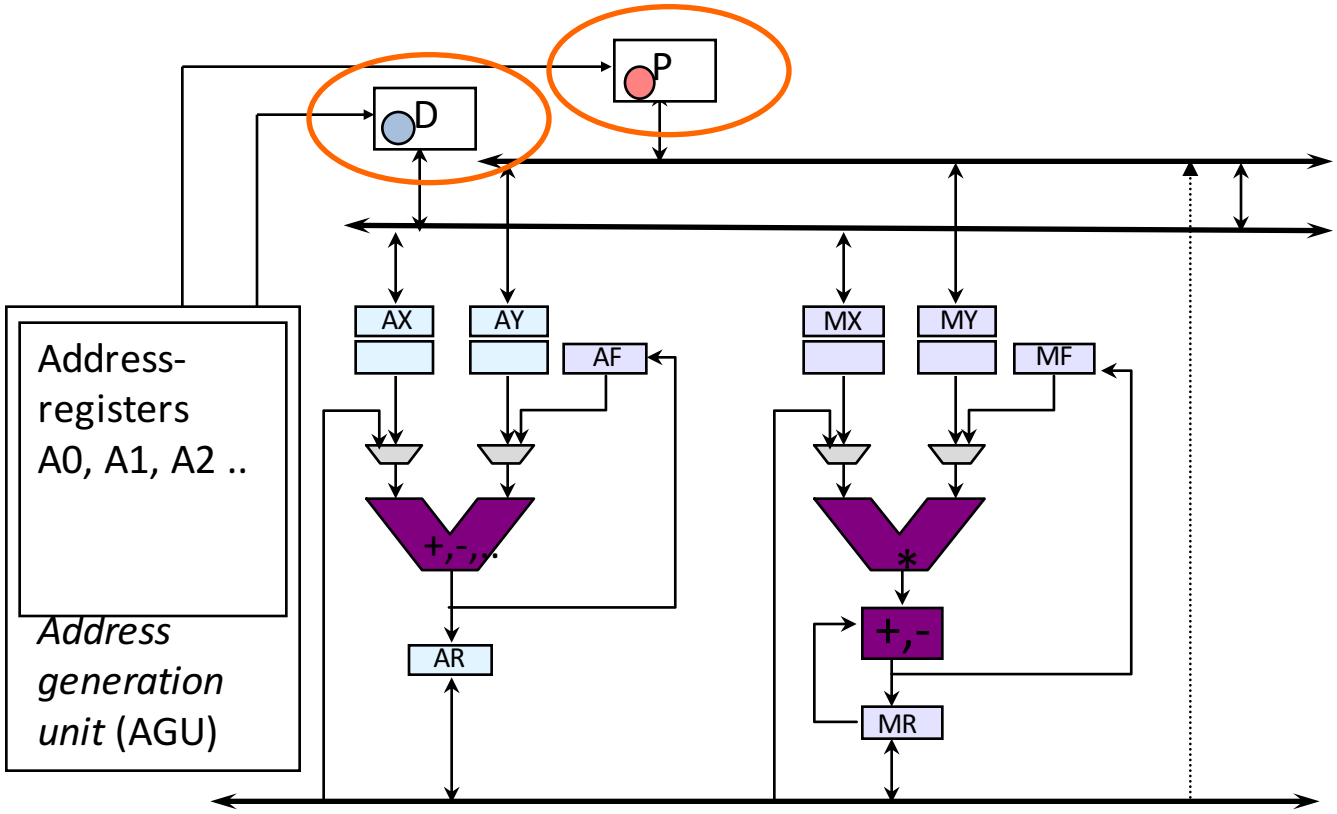
Real-time Processing

- **Timing behavior has to be predictable**
  Features that can cause problems:
  - Unpredictable access to shared resources
    - Caches with difficult to predict replacement strategies
    - Unified caches (conflicts between instructions and data)
    - Pipelines with difficult to predict stall cycles ("bubbles")
    - Unpredictable communication times for multiprocessors
  - Branch prediction, speculative execution
  - Interrupts that are possible any time
  - Memory refreshes (DRAM) that are possible any time
  - Instructions that have data-dependent execution times

⇒ Designers try to avoid as many of these as possible

[Dagstuhl workshop on predictability, Nov. 17-19, 2003]

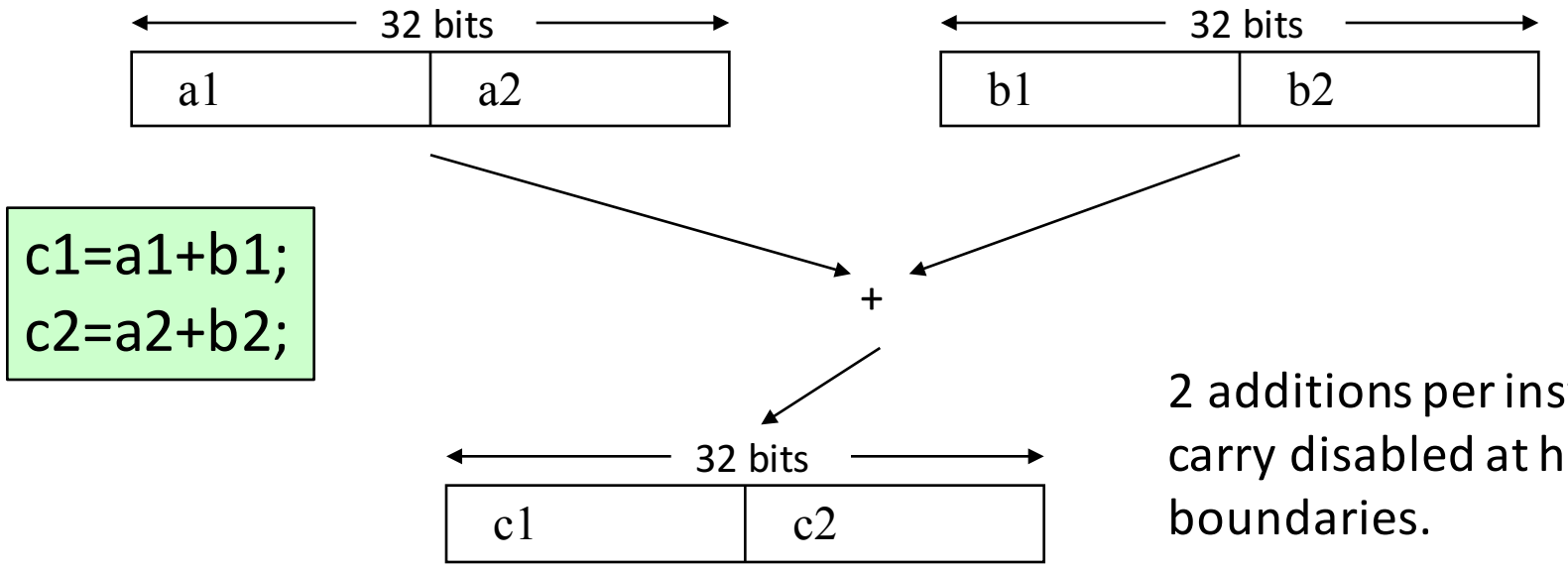
Multiple Memory Banks or Memories



- Simplifies parallel fetches

Multimedia Instructions/Processors

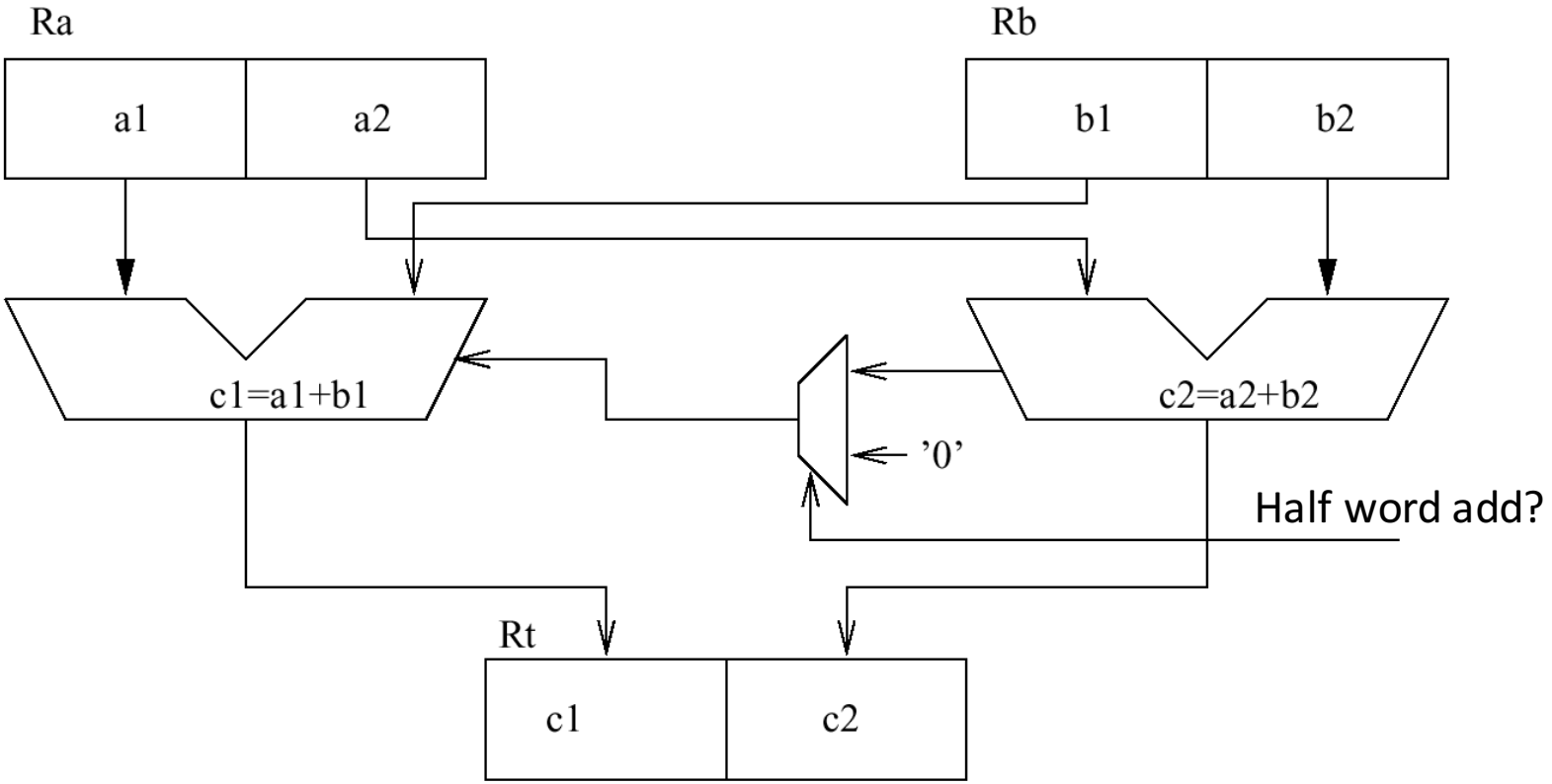
- Most multimedia data types are narrow (*e.g.* 8 bit per color, 16 bit per audio sample per channel)
- Multimedia instructions exploit wide registers and adders (e.g., 32 or 64 bit)
- 2-8 values can be stored and added simultaneously
- Example:



2 additions per instruction; carry disabled at half-word boundaries.

HP *precision architecture* (hp PA)

Half word add instruction **HADD**:



Optional saturating arithmetic

Up to 10 instructions can be replaced by **HADD**

Pentium MMX Architecture (1)

- Multimedia (floating point) registers mm0-mm7
- 64-bit vectors can represent
  - 8 byte encoded,
  - 4 word encoded, or
  - 2 double word encoded numbers.
- Options specify wrap around/saturating arithmetic

Instruction	Options	Comments
Padd[b/w/d] PSub[b/w/d]	*wrap around,* *saturating*	addition/subtraction of bytes, words, double words
Pcmpeq[b/w/d] Pcmpgt[b/w/d]		Result= "11..11" if true, "00..00" otherwise Result= "11..11" if true, "00..00" otherwise
Pmullw Pmulhw		multiplication, 4*16 bits, least significant word multiplication, 4*16 bits, most significant word

Pentium MMX Architecture (2)

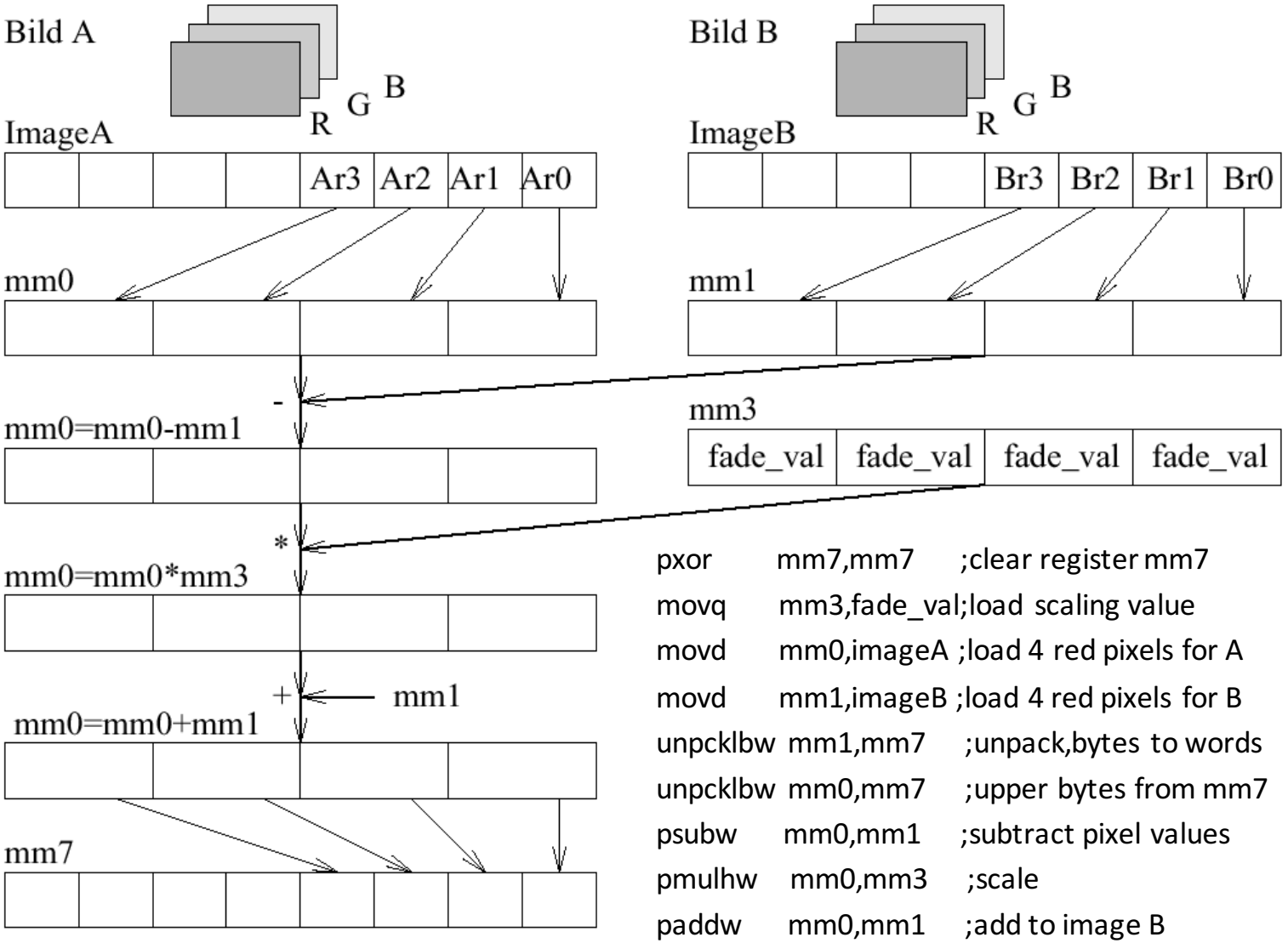
Psra[w/d] Psll[w/d/q] Psrl[w/d/q]	No. of positions in register or instruction	Parallel shift of words, double words or 64 bit quad words
Punpckl[bw/wd/dq] Punpckh[bw/wd/dq]		Parallel unpack Parallel unpack
Packss[wb/dw]	*saturating*	Parallel pack
Pand, Pandn Por, Pxor		Logical operations on 64 bit words
Mov[d/q]		Move instruction

Application of MMX

Scaled interpolation of two images

Next word = next pixel, same color

4 pixels processed at a time




```
pxor       mm7,mm7      ;clear register mm7
movq       mm3,fade_val;load scaling value
movd       mm0,imageA ;load 4 red pixels for A
movd       mm1,imageB ;load 4 red pixels for B
unpcklbw  mm1,mm7      ;unpack,bytes to words
unpcklbw  mm0,mm7      ;upper bytes from mm7
psubw      mm0,mm1      ;subtract pixel values
pmulhw     mm0,mm3      ;scale
paddw      mm0,mm1      ;add to image B
packuswb  mm0,mm7      ;pack words to bytes
```

Short vector instruction set extensions

- 3DNow! (AMD, 1989)
- Streaming SIMD Extensions SSE (Intel, 1999)
  - 16 new registers, floating point SIMD
- SSE2 (Intel, 2001; AMD, 2003)
  - MMX instructions available for new SSE registers
- SSE3 (Intel, 2004; AMD)
  - vector reduction, floating point conversion independent of global rounding mode, relaxed alignment restrictions
- SSE4 (Intel, 2006; AMD: 4 instructions implemented)
  - String comparison, counting 1's, CRC, …
- SSE5 (AMD, 2007)
  - 3-address instructions, …
- Advanced vector extensions AVX (Intel, 2008)
  - Registers 256, … bit wide

Summary

- Embedded systems employ hardware in a loop
- Information processing
  - Importance of energy efficiency
  - Special purpose HW (ASICs) are very expensive
  - Energy efficiency of processors
  - Code-size efficiency
  - Run-time efficiency

Next Time

- Embedded Information Processing
  - Very Long Instruction Word (VLIW) Processors
  - Reconfigurable Computing
  - Chapter 3.3
- Embedded Storage
  - Chapter 3.4