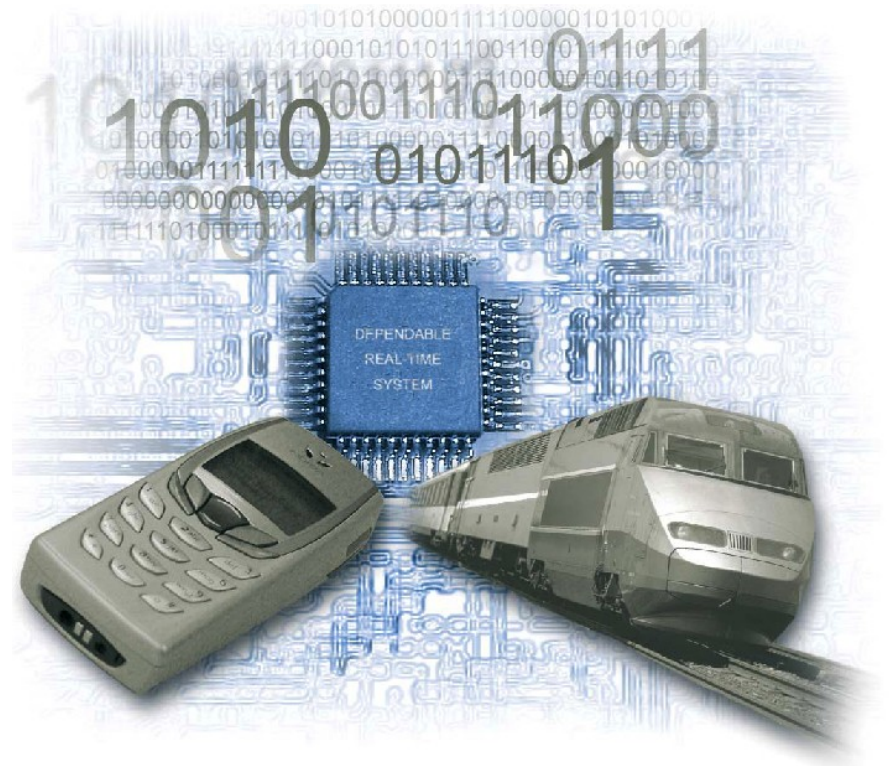


Embedded System Hardware - Processing -

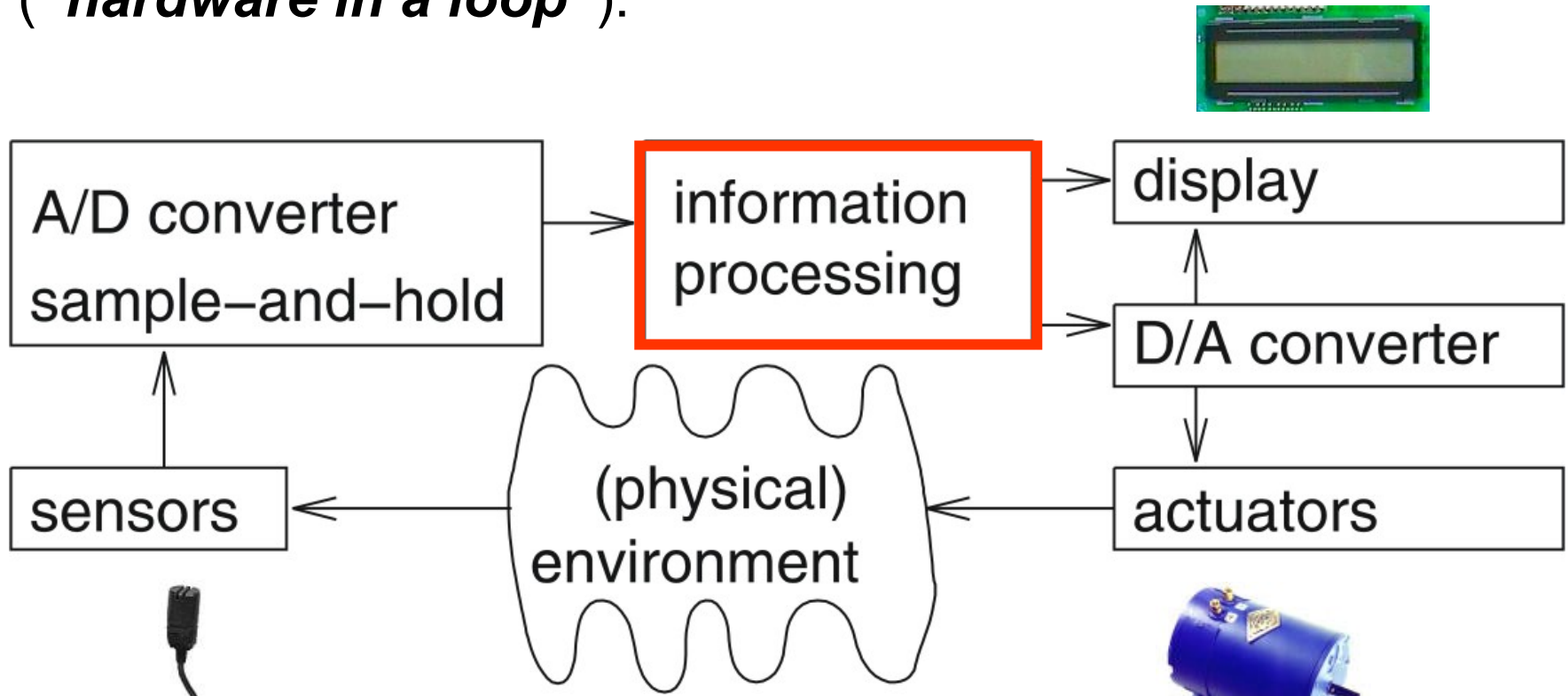
Peter Marwedel
Informatik 12
TU Dortmund
Germany

2009/11/17



Embedded System Hardware

Embedded system hardware is frequently used in a loop (*“hardware in a loop”*):

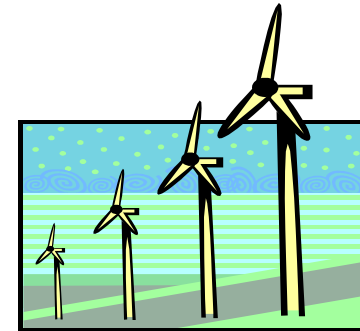


👉 cyber-physical systems

Processing units

Need for efficiency (power + energy):

Why worry about energy and power?



“Power is considered as the most important constraint in embedded systems“

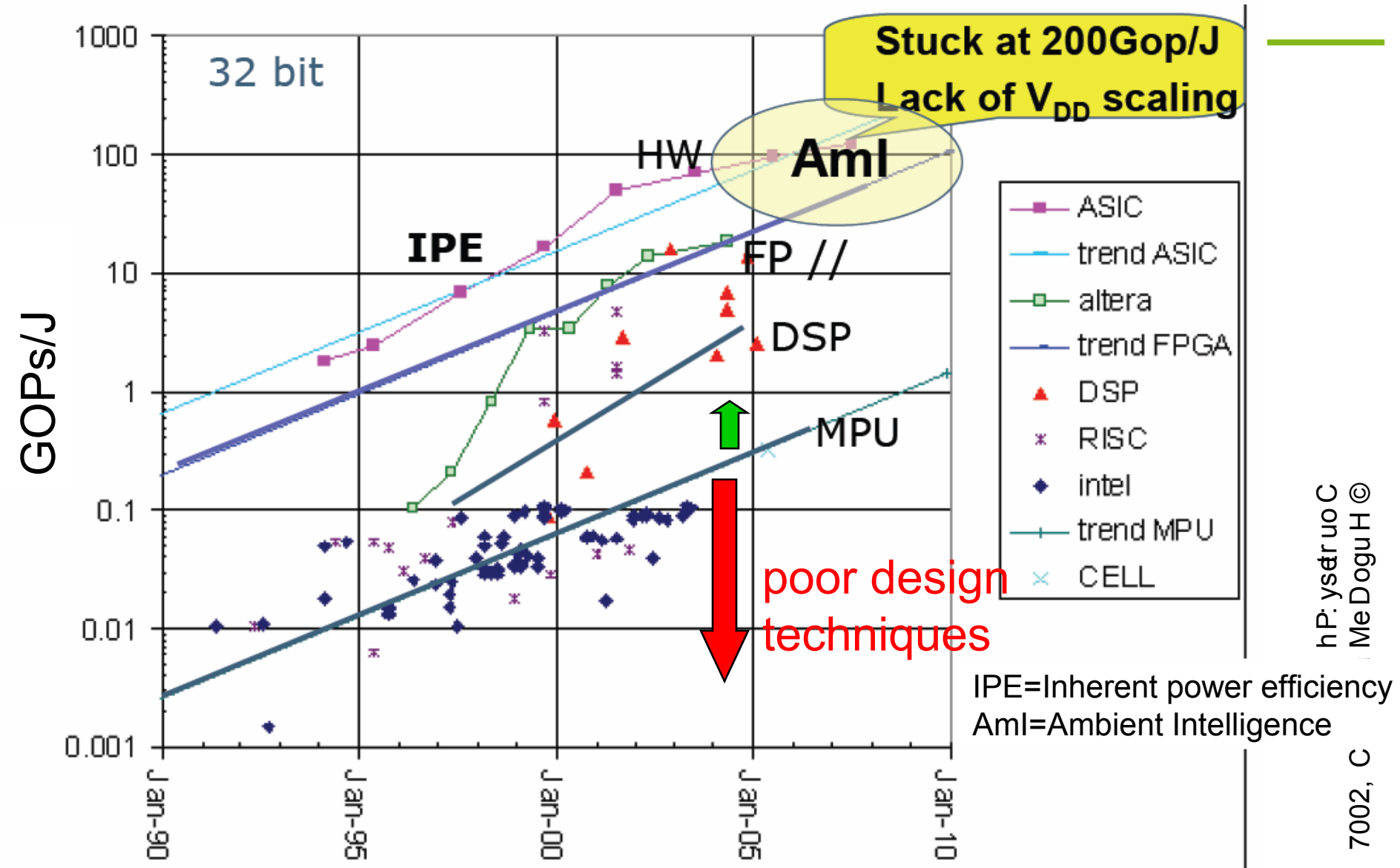
[in: L. Eggermont (ed): Embedded Systems Roadmap 2002, STW]

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



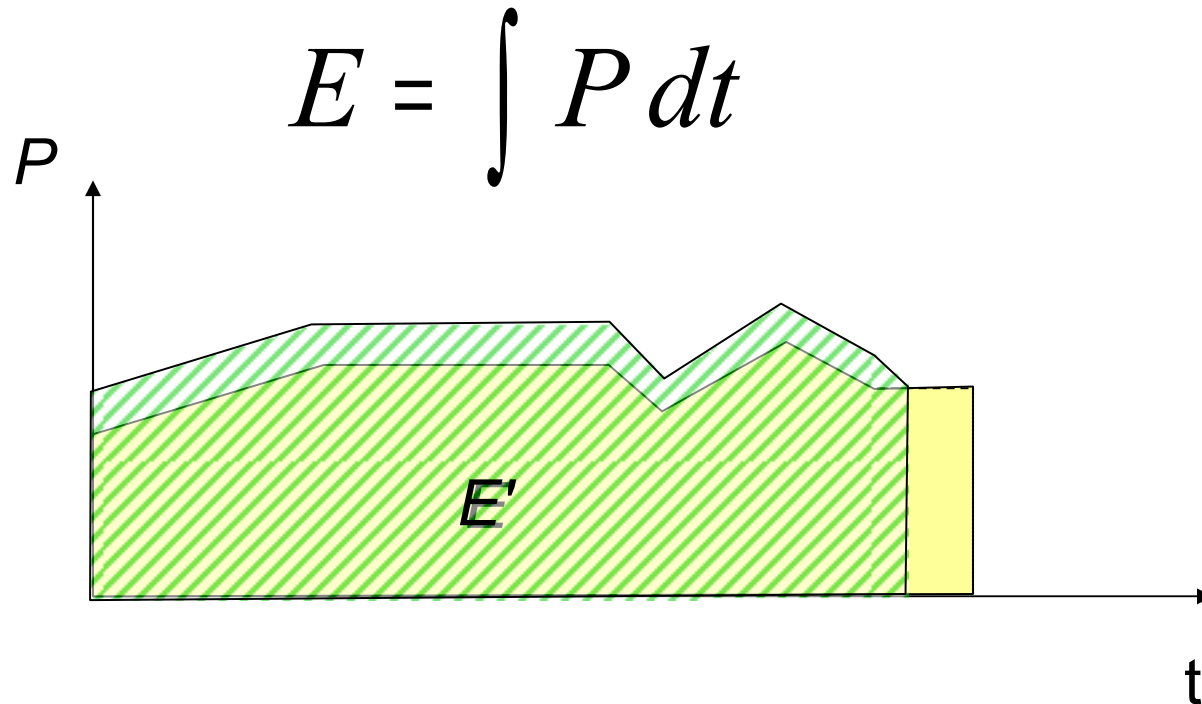
<http://www.esa.int/images/earth,4.jpg>

Importance of Energy Efficiency



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

Power and energy are related to each other



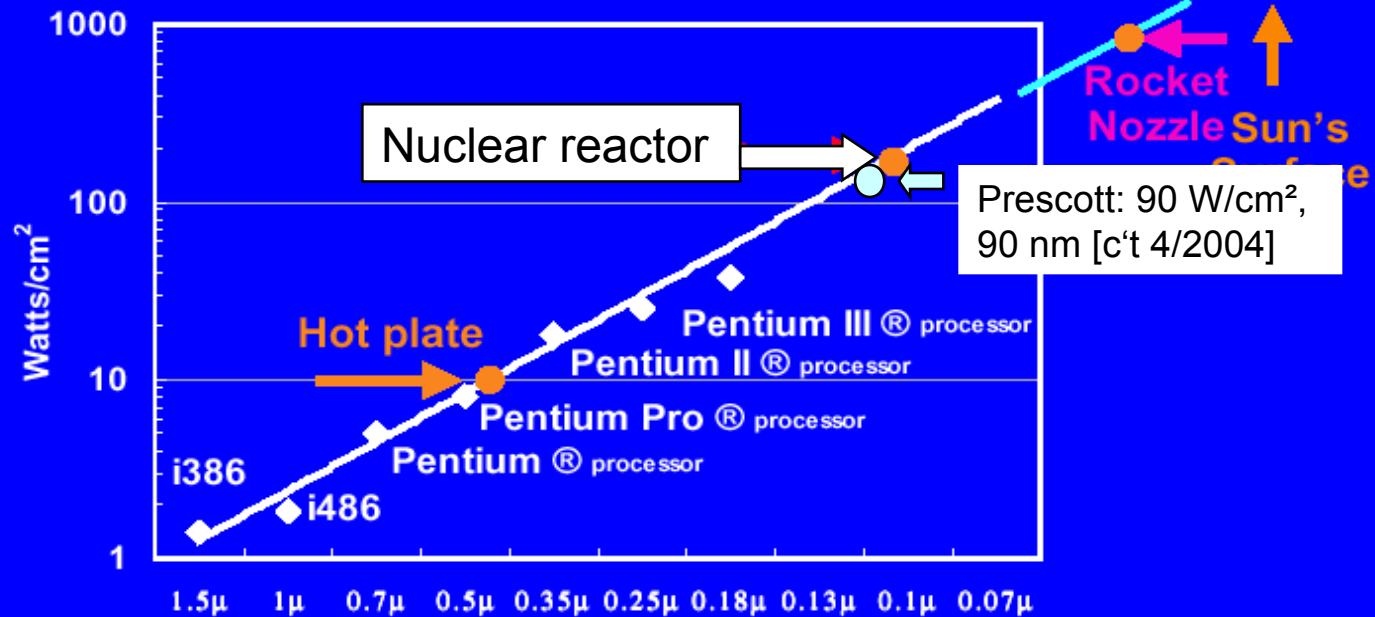
In many cases, faster execution also means less energy, but the opposite may be true if power has to be increased to allow faster execution.

Low Power vs. Low Energy Consumption

- Minimizing the **power consumption** is important for
 - the design of the power supply
 - the design of voltage regulators
 - the dimensioning of interconnect
 - short term cooling
- Minimizing the **energy consumption** is 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
 - dependability
 - long lifetimes, low temperatures

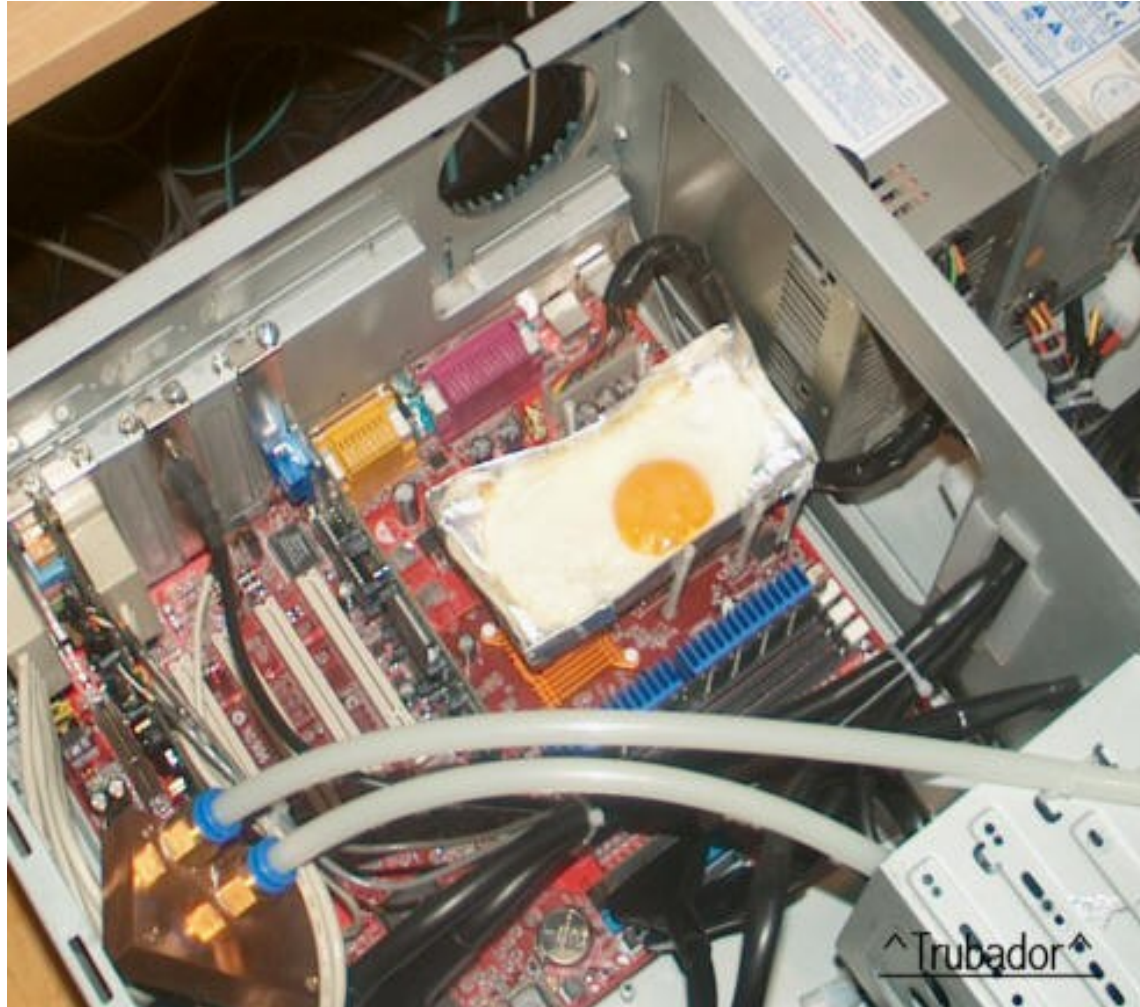


Power density continues to get worse



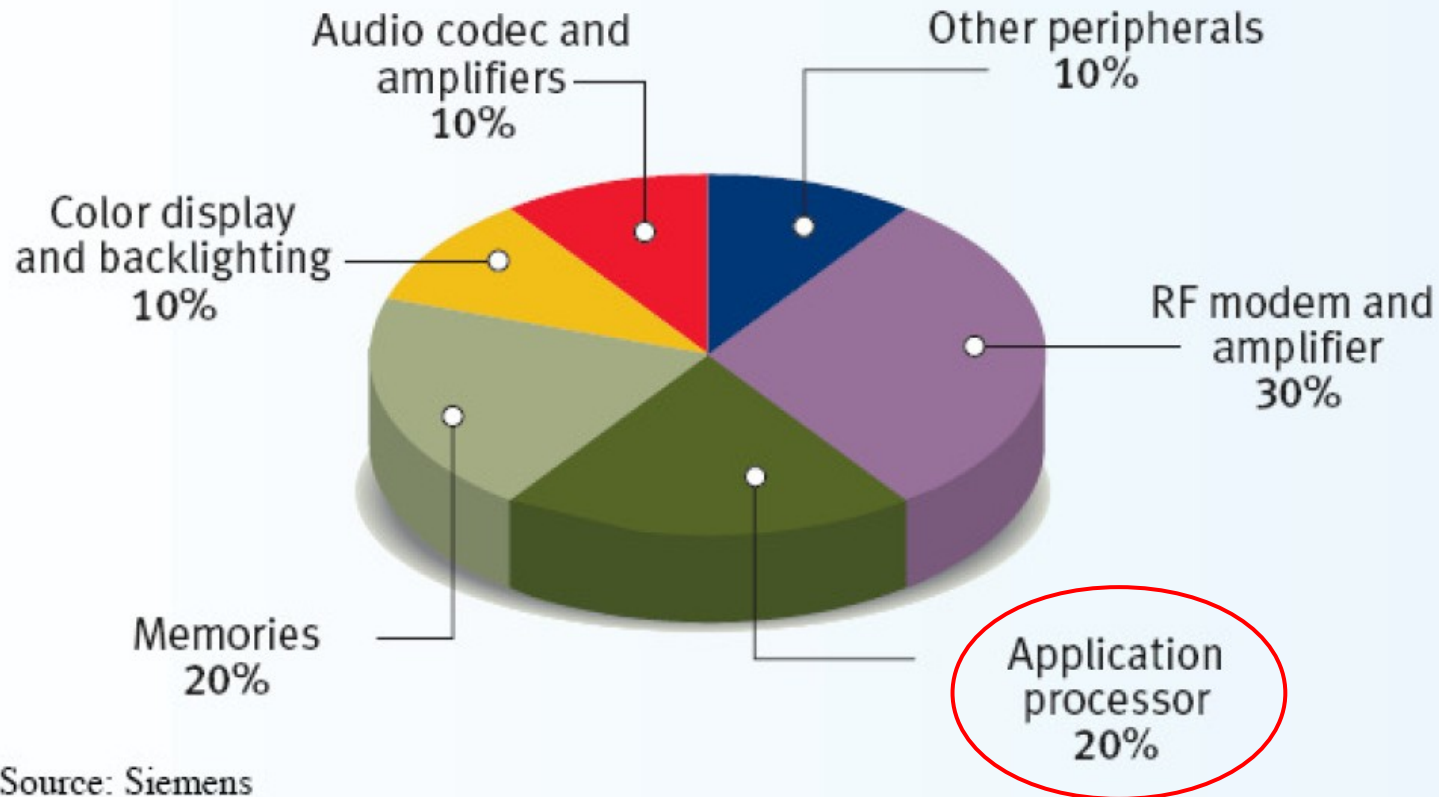
Surpassed hot-plate power density in 0.5μ
Not too long to reach nuclear reactor

Surpassed hot (kitchen) plate ...? Why not use it?



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

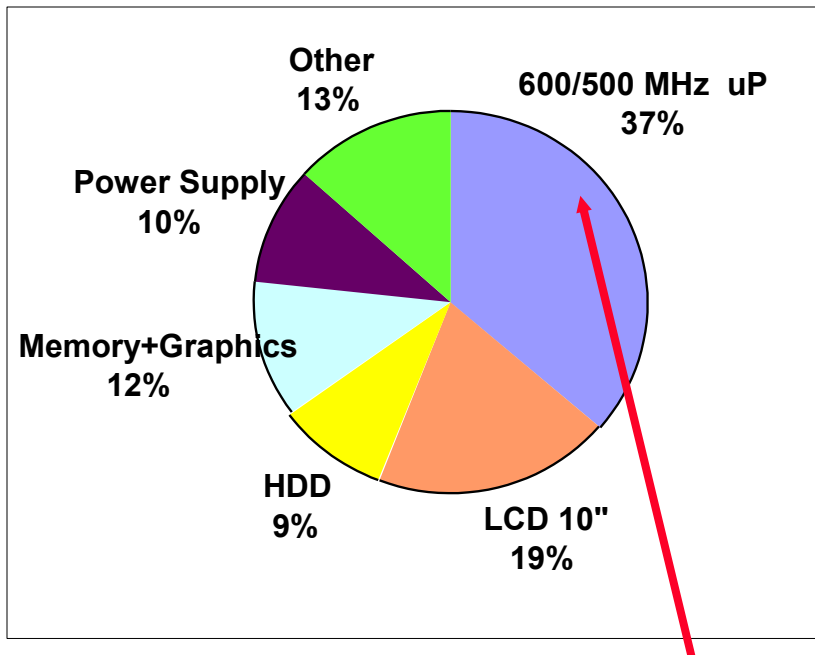
Energy consumption in mobile devices



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

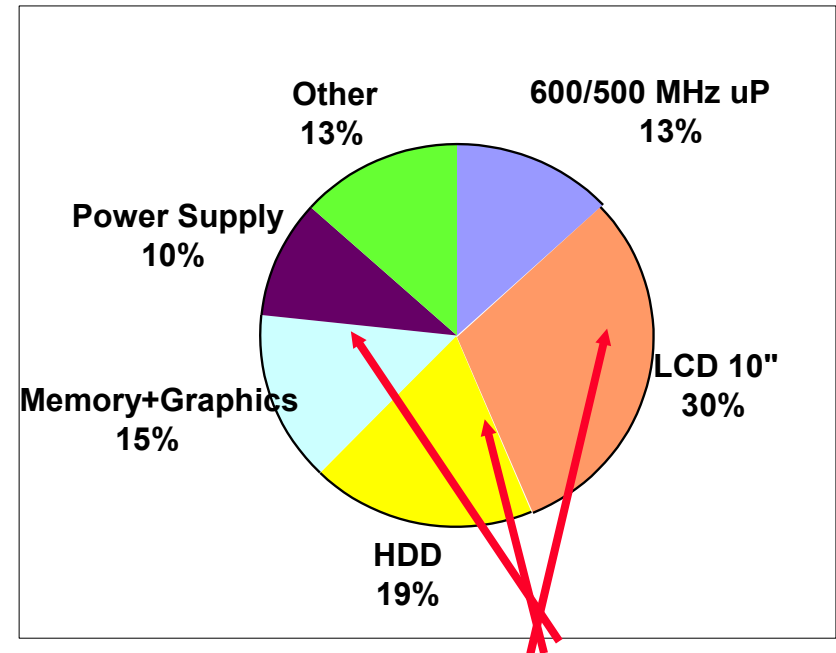
Need to consider CPU & System Power

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



CPU Dominates Thermal Design Power

**Mobile PC (notebook)
Average System Power**



Note: Based on Actual Measurements

Multiple Platform Components Comprise Average Power

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

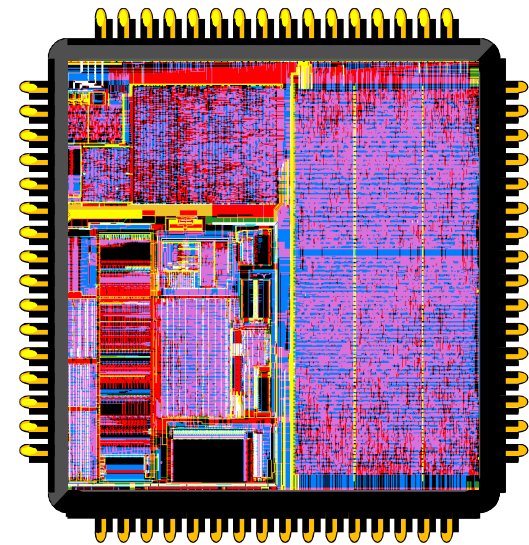
Application Specific Circuits (ASICs) or Full Custom Circuits

Custom-designed circuits necessary

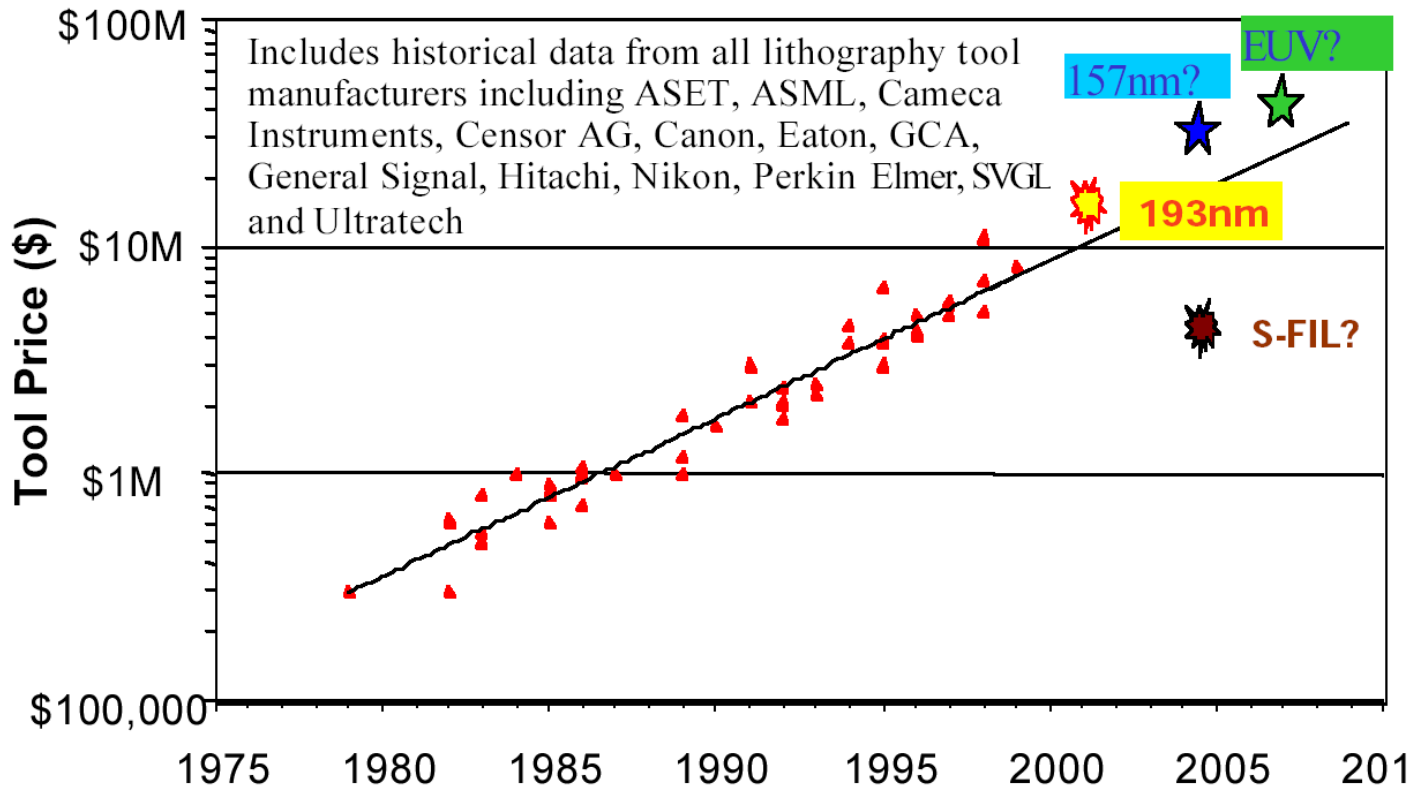
- if ultimate speed or
- energy efficiency is the goal and
- large numbers can be sold.

Approach suffers from

- long design times,
- lack of flexibility
(changing standards) and
- high costs
(e.g. Mill. \$ mask costs).



Mask cost for specialized HW becomes very expensive



➔ Trend towards implementation in Software

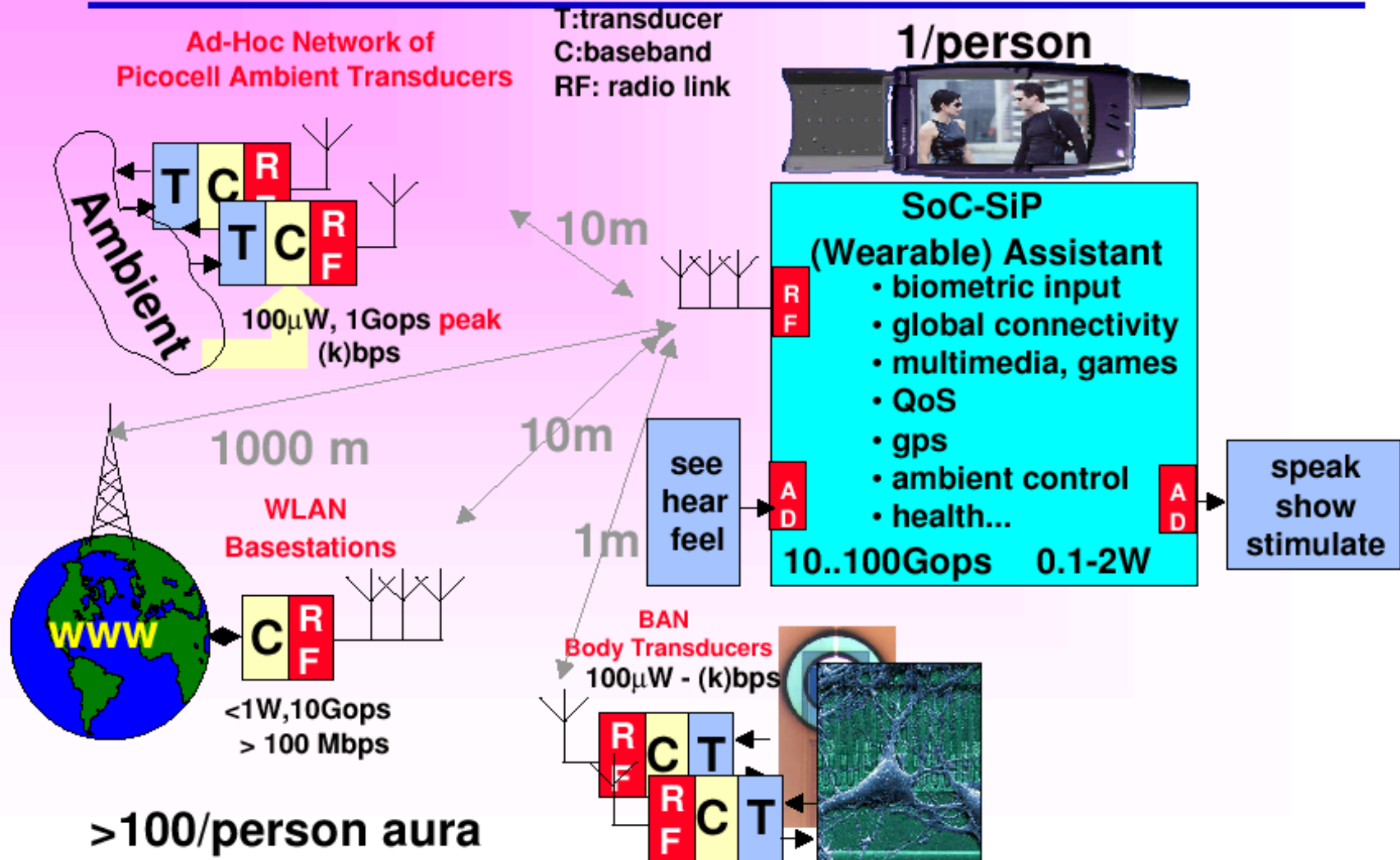
HW synthesis not covered in this course.

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

Key requirements for processors

1. Energy/power-efficiency

Ambient Intelligence Global System



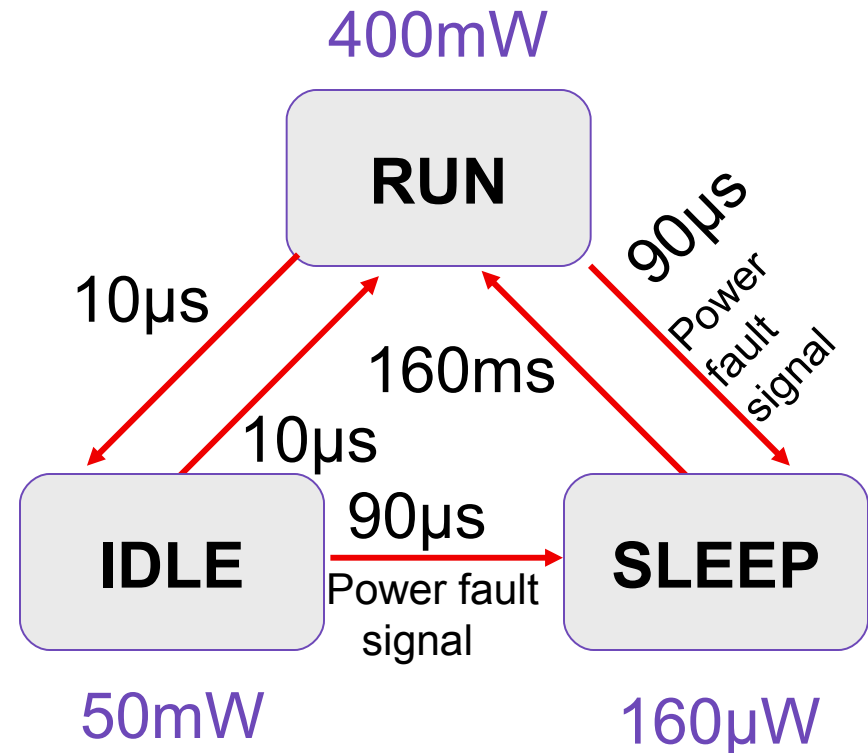
Dynamic power management (DPM)

Example: STRONGARM SA1100

RUN: operational

IDLE: a sw routine may stop the CPU when not in use, while monitoring interrupts

SLEEP: Shutdown of on-chip activity



Fundamentals of dynamic voltage scaling (DVS)

Power consumption of CMOS circuits (ignoring leakage):

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

α : switching activity

C_L : load capacitance

V_{dd} : supply voltage

f : clock frequency

Delay for CMOS circuits:

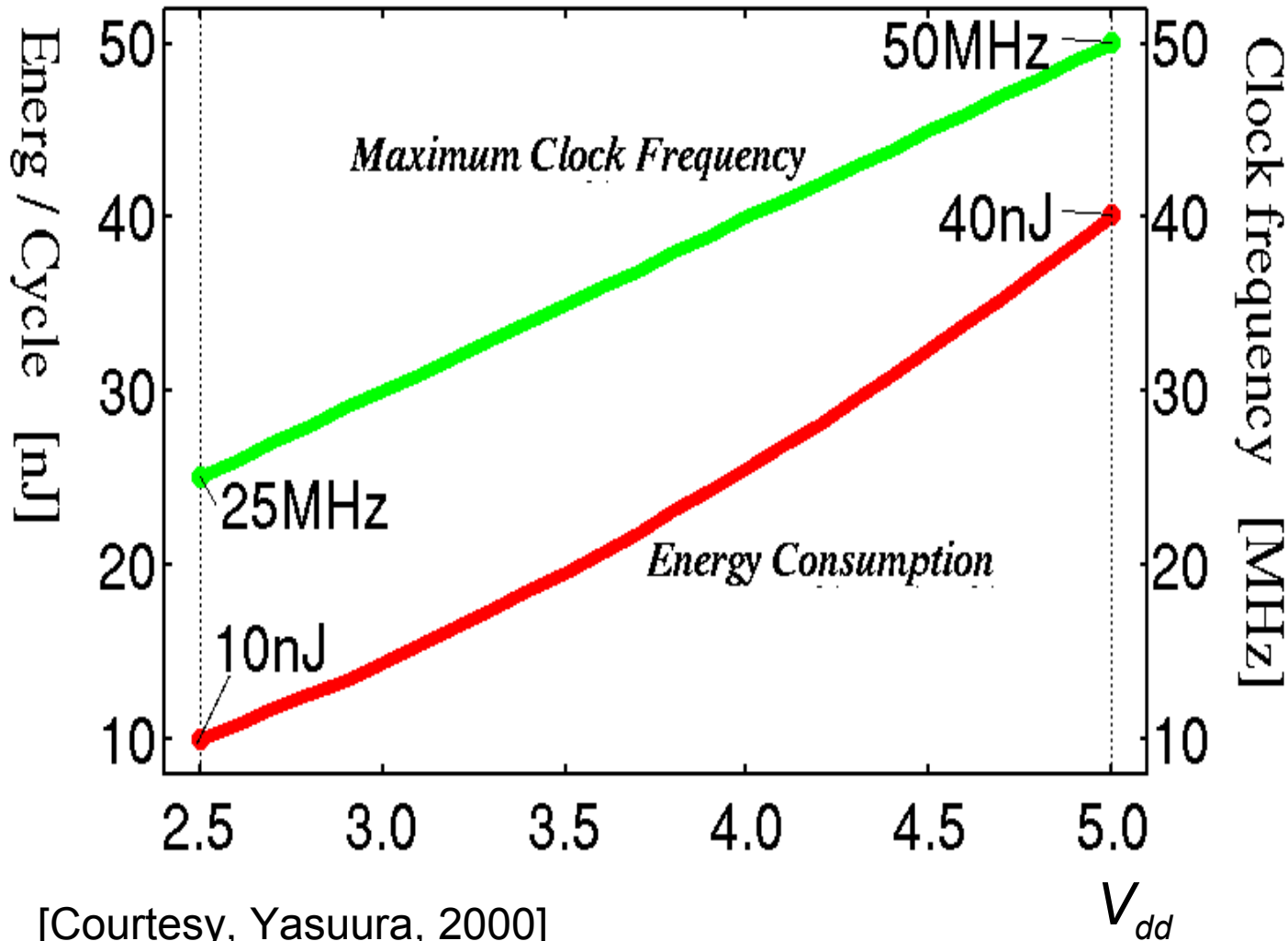
$$\tau = k C_L \frac{V_{dd}}{(V_{dd} - V_t)^2} \text{ with}$$

V_t : threshold voltage

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

☞ Decreasing V_{dd} reduces P quadratically,
while the run-time of algorithms is only linearly increased
 $E = P \times t$ decreases linearly
(ignoring the effects of the memory system and V_t)

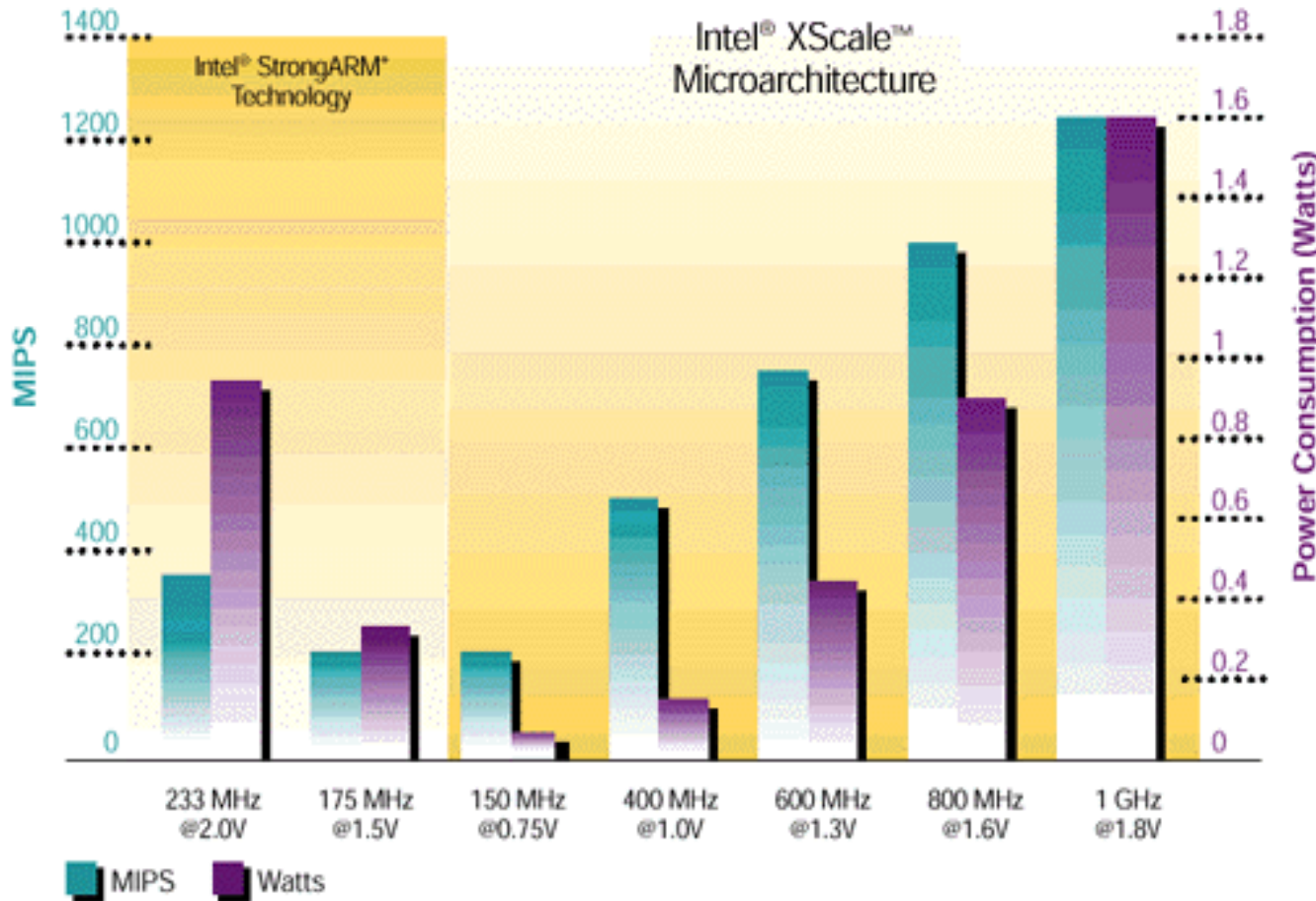
Voltage scaling: Example



[Courtesy, Yasuura, 2000]

Variable-voltage/frequency example: INTEL Xscale

POWER-PERFORMANCE COMPARISON



OS should schedule distribution of the energy budget.

Application: VLIW procesing and voltage scaling in the Crusoe processor

- V_{DD} : 32 levels (1.1V - 1.6V)
- Clock: 200MHz - 700MHz in increments of 33MHz

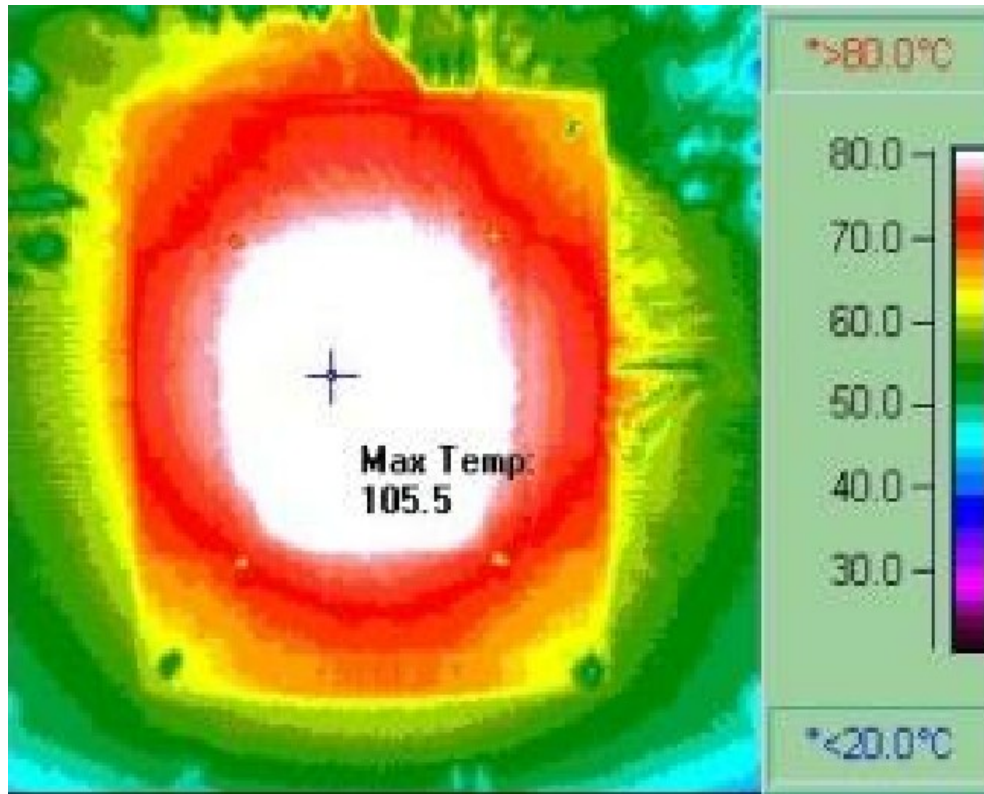
Scaling is triggered when CPU load change is detected by software ($\sim 1/2$ ms).

- More load: Increase of supply voltage (~ 20 ms/step), followed by scaling clock frequency
- Less load: reduction of clock frequency, followed by reduction of supply voltage

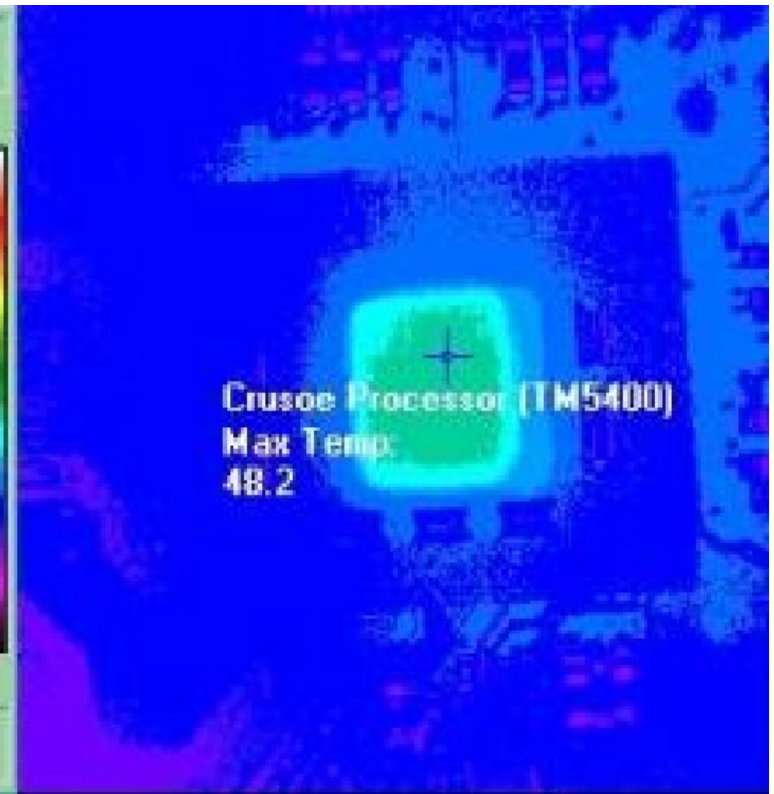
Worst case (1.1V to 1.6V V_{DD} , 200MHz to 700MHz) takes 280 ms

Result (as published by transmeta)

Pentium



Crusoe

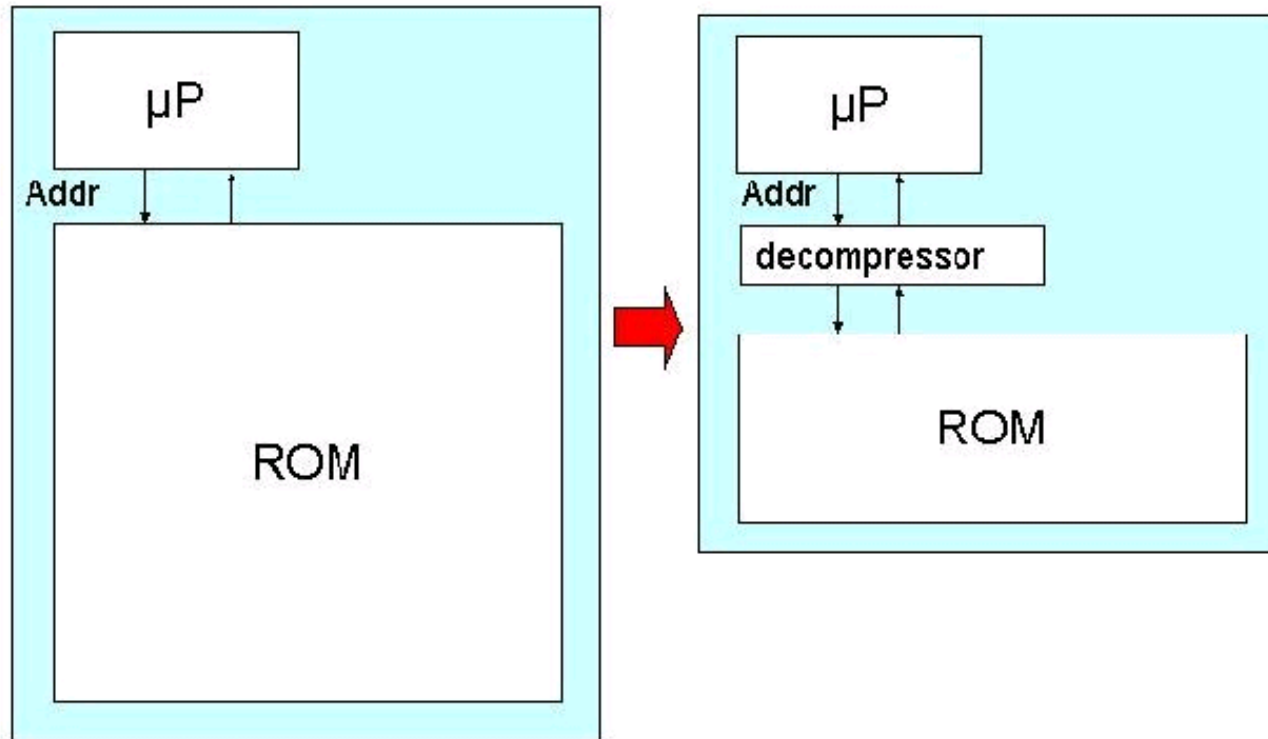


Running the same multimedia application.

[www.transmeta.com]

Key requirement #2: Code-size efficiency

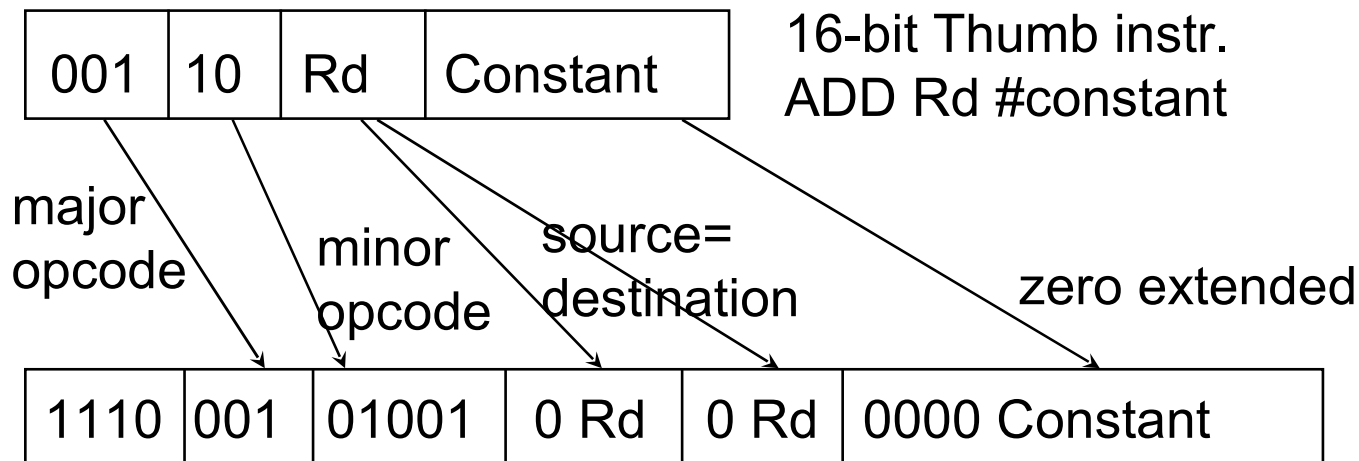
- **CISC machines:** RISC machines designed for run-time-, not for code-size-efficiency
- **Compression techniques:** key idea



Code-size efficiency

■ Compression techniques (continued):

- 2nd instruction set, e.g. ARM Thumb instruction set:



Dynamically
decoded at
run-time

- 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.

Dictionary approach, two level control store (indirect addressing of instructions)

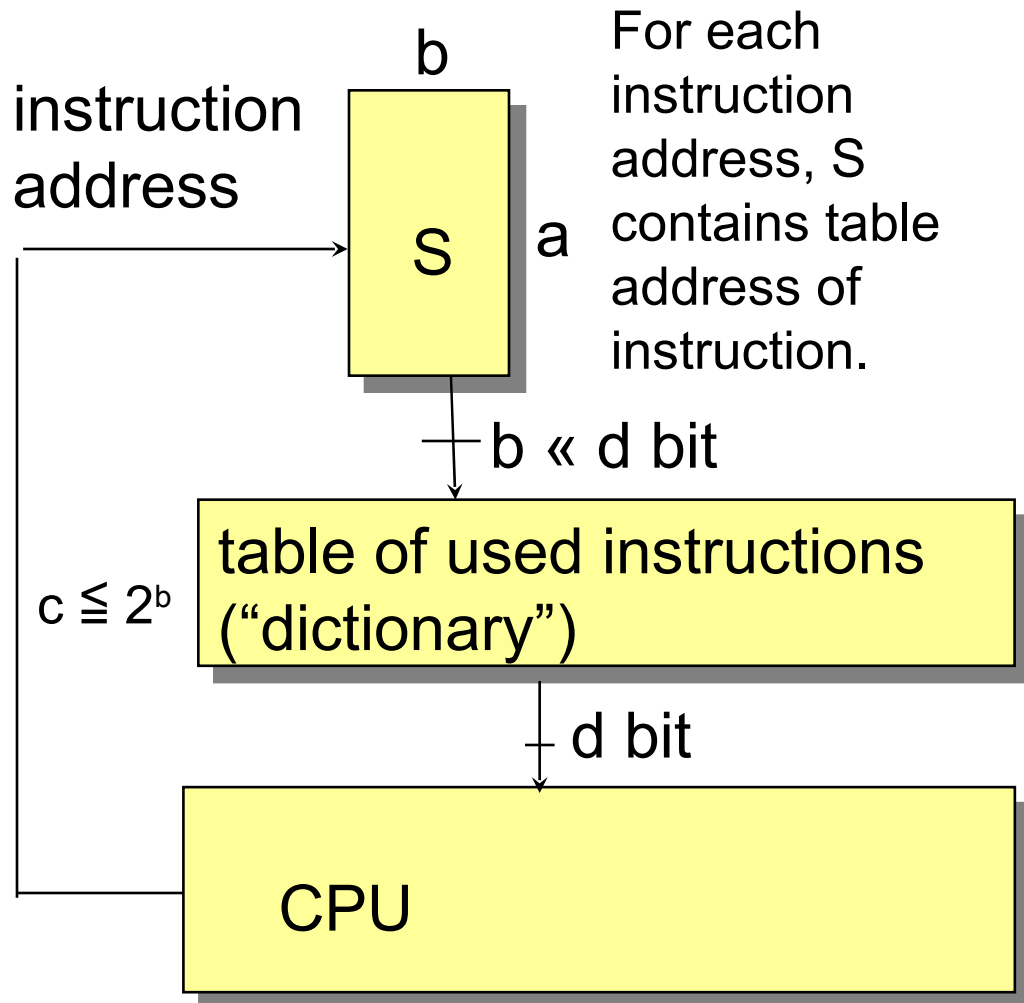
“Dictionary-based coding schemes cover a wide range of various coders and compressors.

Their common feature is that the methods 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. Beszédés et al.: Survey of Code size Reduction Methods, Survey of Code-Size Reduction Methods, *ACM Computing Surveys*, Vol. 35, Sept. 2003, pp 223-267]

Key idea (for d bit instructions)



Uncompressed storage of a d -bit-wide instructions requires axd bits.

In compressed code, each instruction pattern is stored only once.

Hopefully, $axb + cxd < axd$.

Called nanoprogramming in the Motorola 68000.

small

Cache-based decompression

- Main idea: decompression whenever cache-lines are fetched from memory.
- Cache lines \leftrightarrow 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 information on code compaction

- 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-perso.iro.umontreal.ca/~latendre/codeCompression/codeCompression/node1.html)

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

(153 entries as per 11/2004)

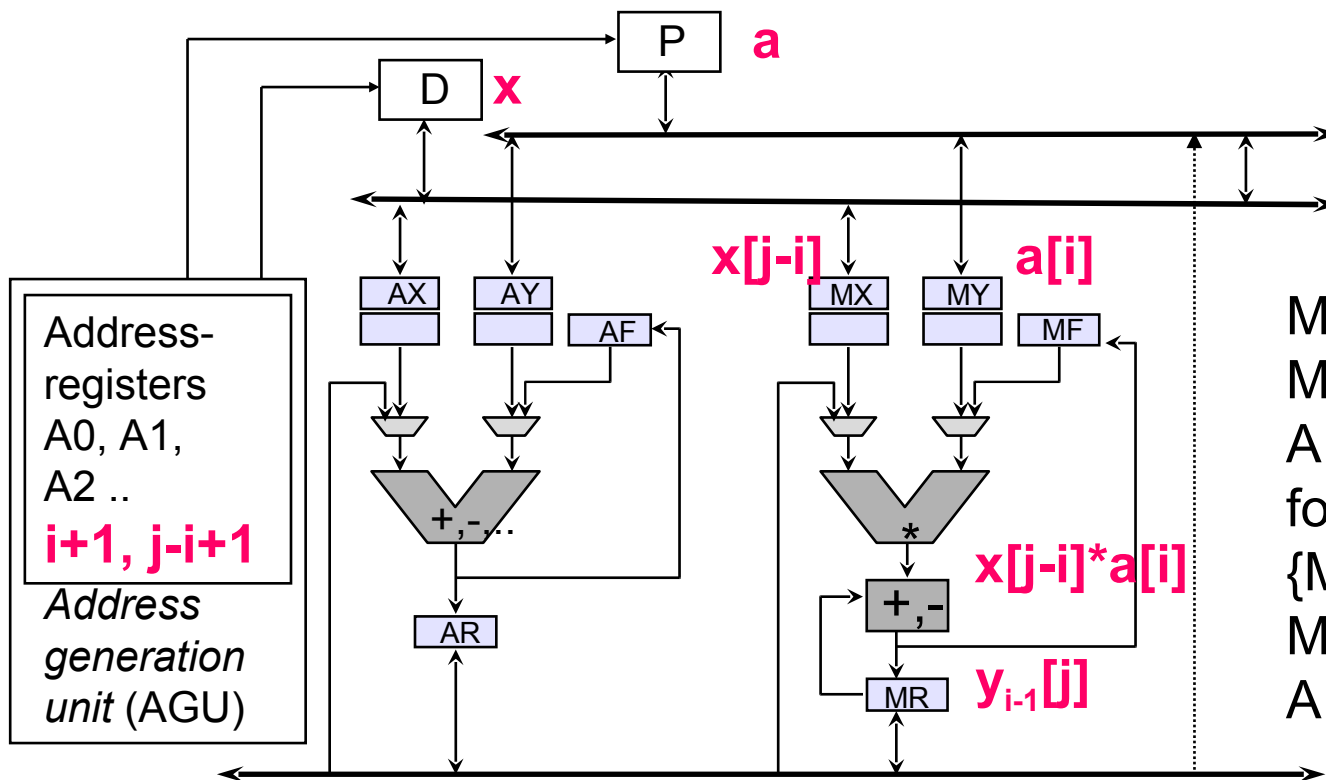
Key requirement #3: Run-time efficiency

- Domain-oriented architectures -

Application: $y[j] = \sum_{i=0}^{n-1} x[j-i] * a[i]$

$\forall i: 0 \leq i \leq n-1: y_i[j] = y_{i-1}[j] + x[j-i] * a[i]$

Architecture: Example: Data path ADSP210x



Application maps nicely onto architecture

```
MR:=0;
MX:=x[n-1]; MY:=a[0];
A1:=1; A2:=n-2;
for ( j:=1 to n)
{MR:=MR+MX*MY;
MY:=a[A1]; MX:=x[A2];
A1++; A2--}
```

DSP-Processors: multiply/accumulate (MAC) and zero-overhead loop (ZOL) instructions

```
MR:=0; A1:=1; A2:=n-2; MX:=x[n-1]; MY:=a[0];
```

```
for ( j:=1 to n)
```

```
{MR:=MR+MX*MY; MY:=a[A1]; MX:=x[A2]; A1++; A2--}
```

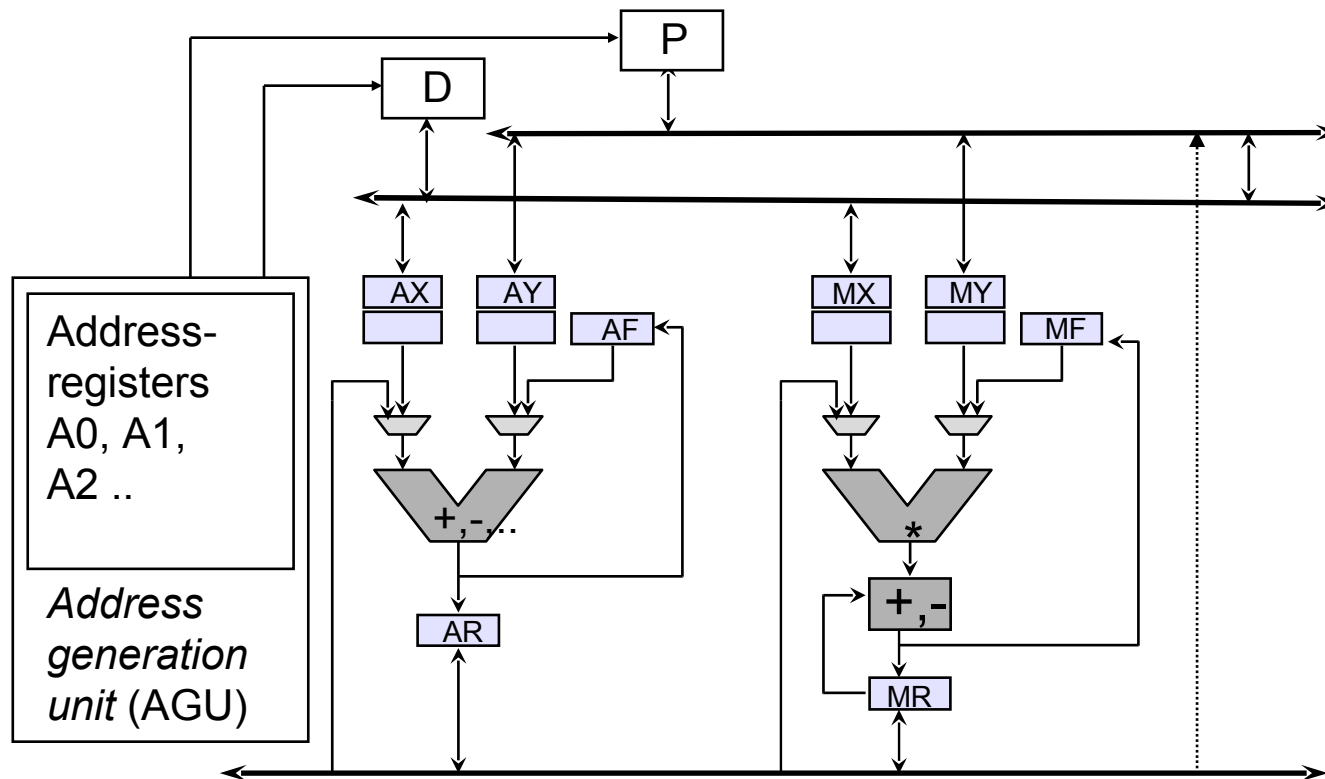
Multiply/accumulate (MAC) instruction

Zero-overhead loop (ZOL) instruction preceding MAC instruction.

Loop testing done in parallel to MAC operations.

Heterogeneous registers

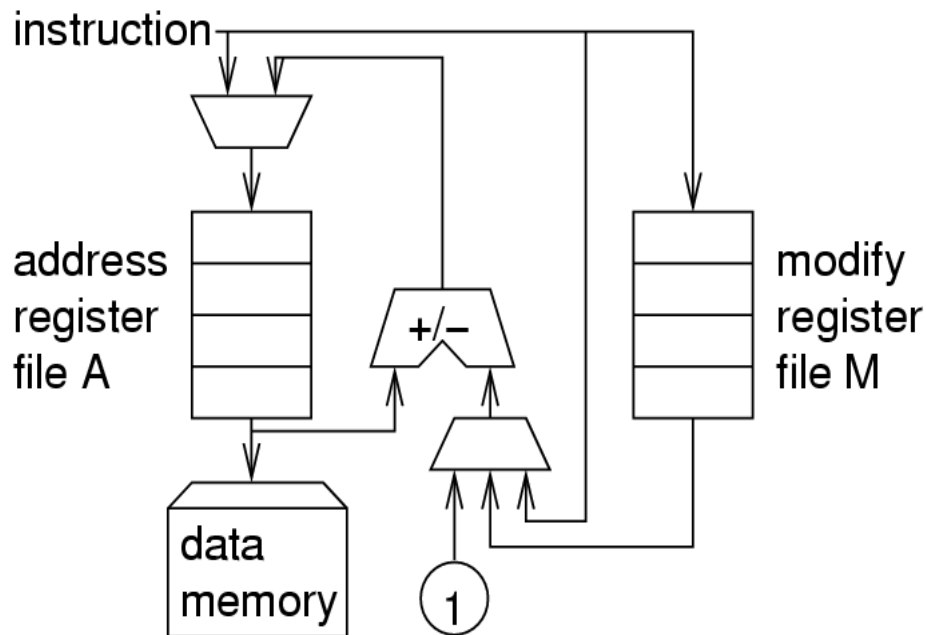
Example (ADSP 210x):



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

Separate address generation units (AGUs)

Example (ADSP 210x):

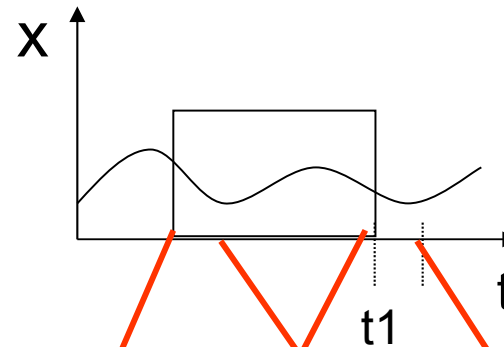


- Data memory can only be fetched with address contained in A,
- but this can be done in parallel with operation in main data path (takes effectively 0 time).
- $A := A \pm 1$ also takes 0 time,
- same for $A := A \pm M$;
- $A := \langle \text{immediate in instruction} \rangle$ requires extra instruction
- ☞ Minimize load immediates
- ☞ Optimization in optimization chapter

Modulo addressing

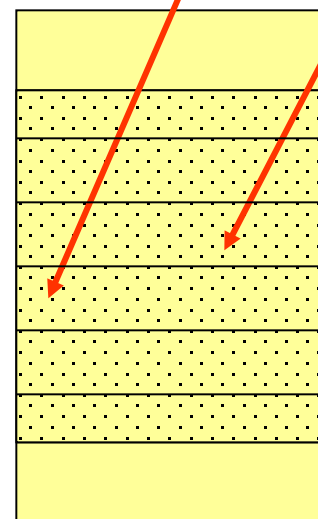
Modulo addressing:
 $A_{m++} \equiv A_m := (A_m + 1) \bmod n$
 (implements ring or circular buffer in memory)

sliding window



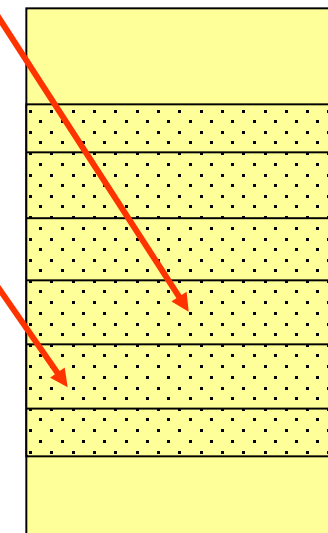
n most recent values

- ..
- $x[t1-1]$
- $x[t1]$
- $x[t1-n+1]$
- $x[t1-n+2]$
- ..



Memory, $t=t1$

- ..
- $x[t1-1]$
- $x[t1]$
- $x[t1+1]$
- $x[t1-n+2]$
- ..



Memory, $t2=t1+1$

Saturating arithmetic

- Returns largest/smallest number in case of over/underflows

- Example:

a		0111
b	+	1001

standard wrap around arithmetic	(1)0000
---------------------------------	---------

saturating arithmetic	1111
-----------------------	------

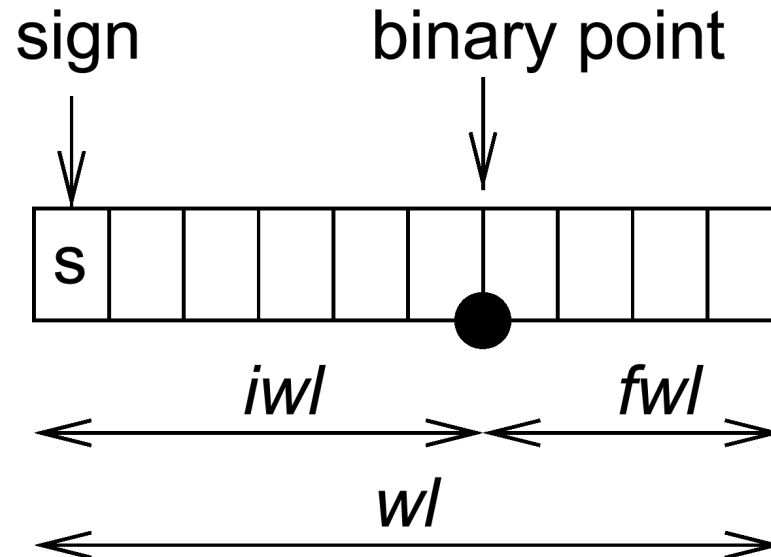
(a+b)/2: correct	1000
-------------------------	------

wrap around arithmetic	0000
------------------------	------

saturating arithmetic + shifted	0111 “almost correct”
---------------------------------	-----------------------

- Appropriate for DSP/multimedia applications:
 - No timeliness of results if interrupts are generated for overflows
 - Precise values 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 a key advantage for multiplications.
- Example:
 $x = 0.5 \times 0.125 + 0.25 \times 0.125 = 0.0625 + 0.03125 = 0.09375$
For $iw/=1$ and $fw/=3$ decimal digits, the less significant digits are automatically chopped off: $x = 0.093$
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).

Real-time capability

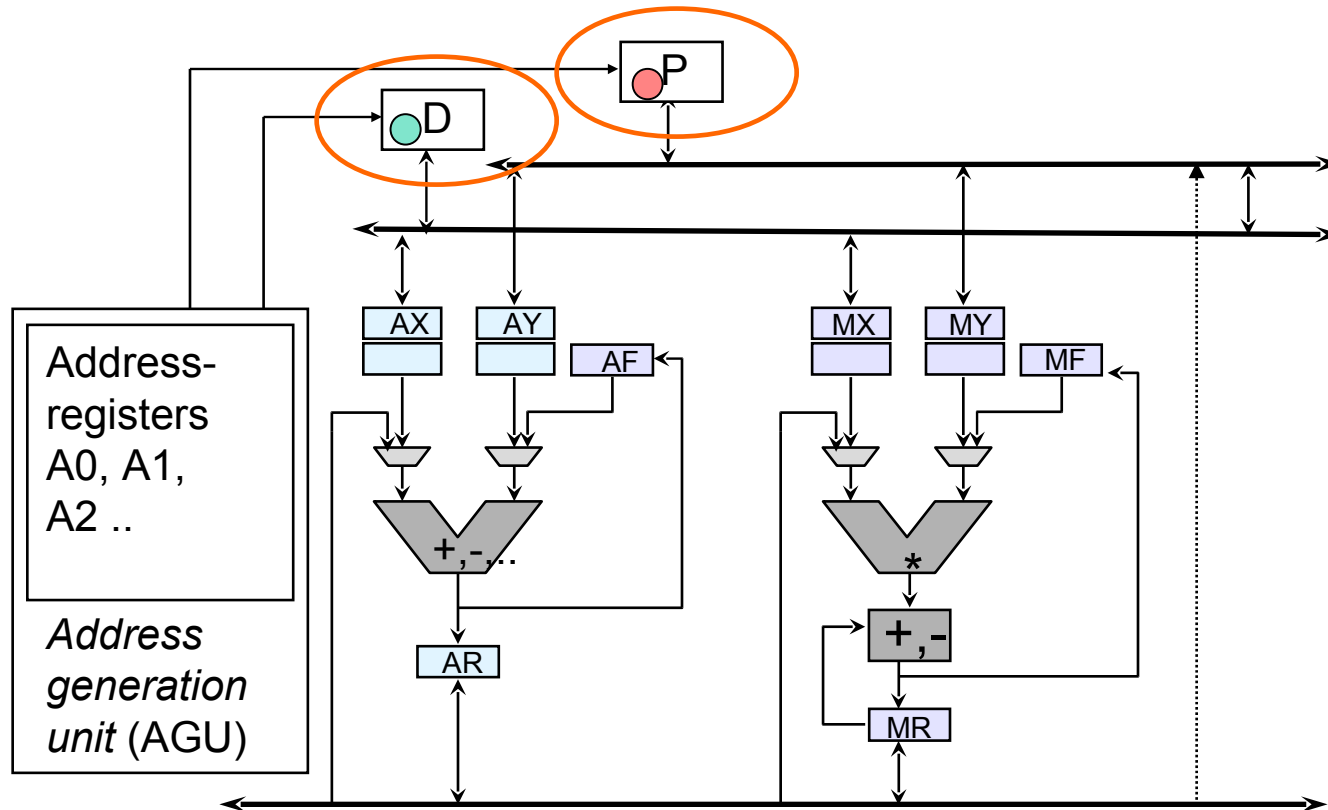
■ Timing behavior has to be predictable

Features that cause problems:

- Unpredictable access to shared resources
 - Caches with difficult to predict replacement strategies
 - Unified caches (conflicts betw. 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 that are possible any time
- Instructions that have data-dependent execution times

 **Trying to avoid as many of these as possible.**

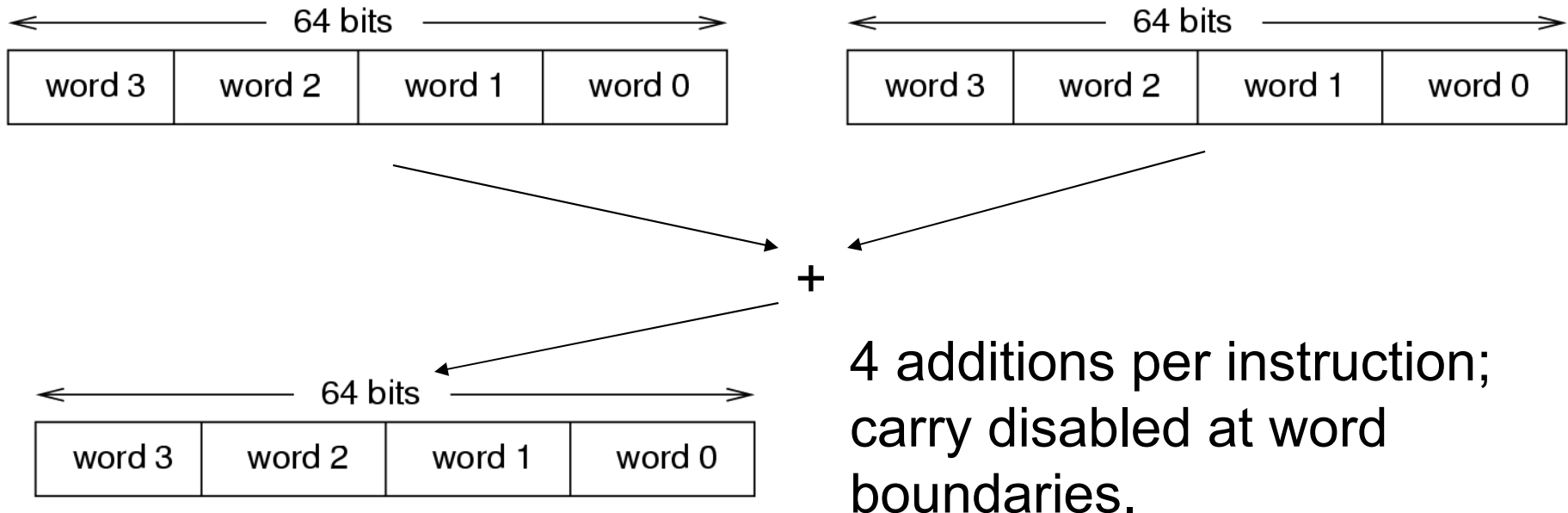
Multiple memory banks or memories



Simplifies parallel fetches

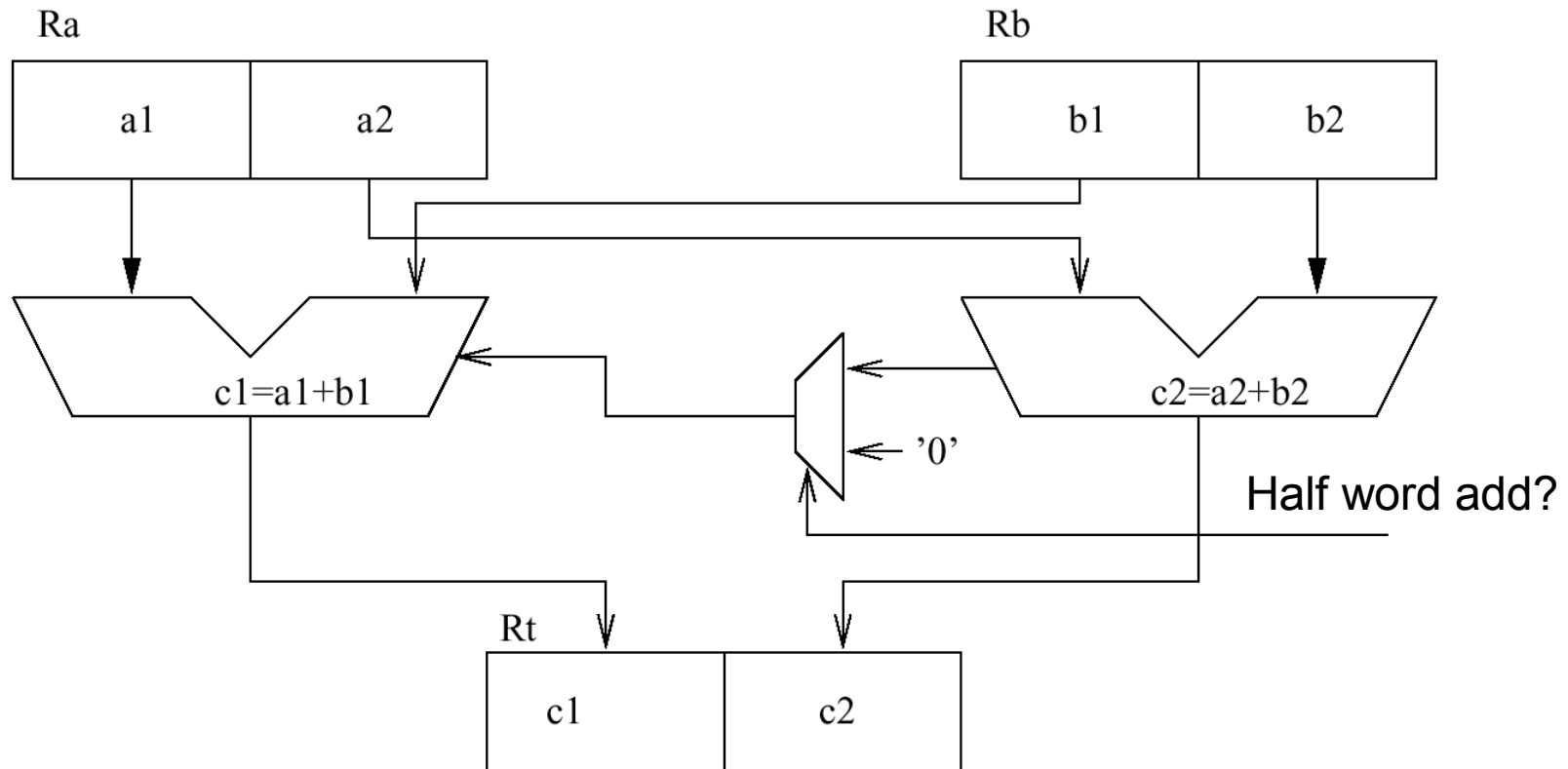
Multimedia-Instructions/Processors

- Multimedia instructions exploit that many registers, adders etc are quite wide (32/64 bit),
- whereas most multimedia data types are narrow (e.g. 8 bit per color, 16 bit per audio sample per channel)
- ☞ 2-8 values can be stored per register and added. E.g.:



Early example: 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)

64-bit vectors representing 8 byte encoded, 4 word encoded or 2 double word encoded numbers.

wrap around/saturating options.

Multimedia registers mm0 - mm7,
consistent with floating-point registers (OS unchanged).

Instruction	Options	Comments
Padd[b/w/d] PSub[b/w/d]	<i>wrap around,</i> <i>saturating</i>	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)

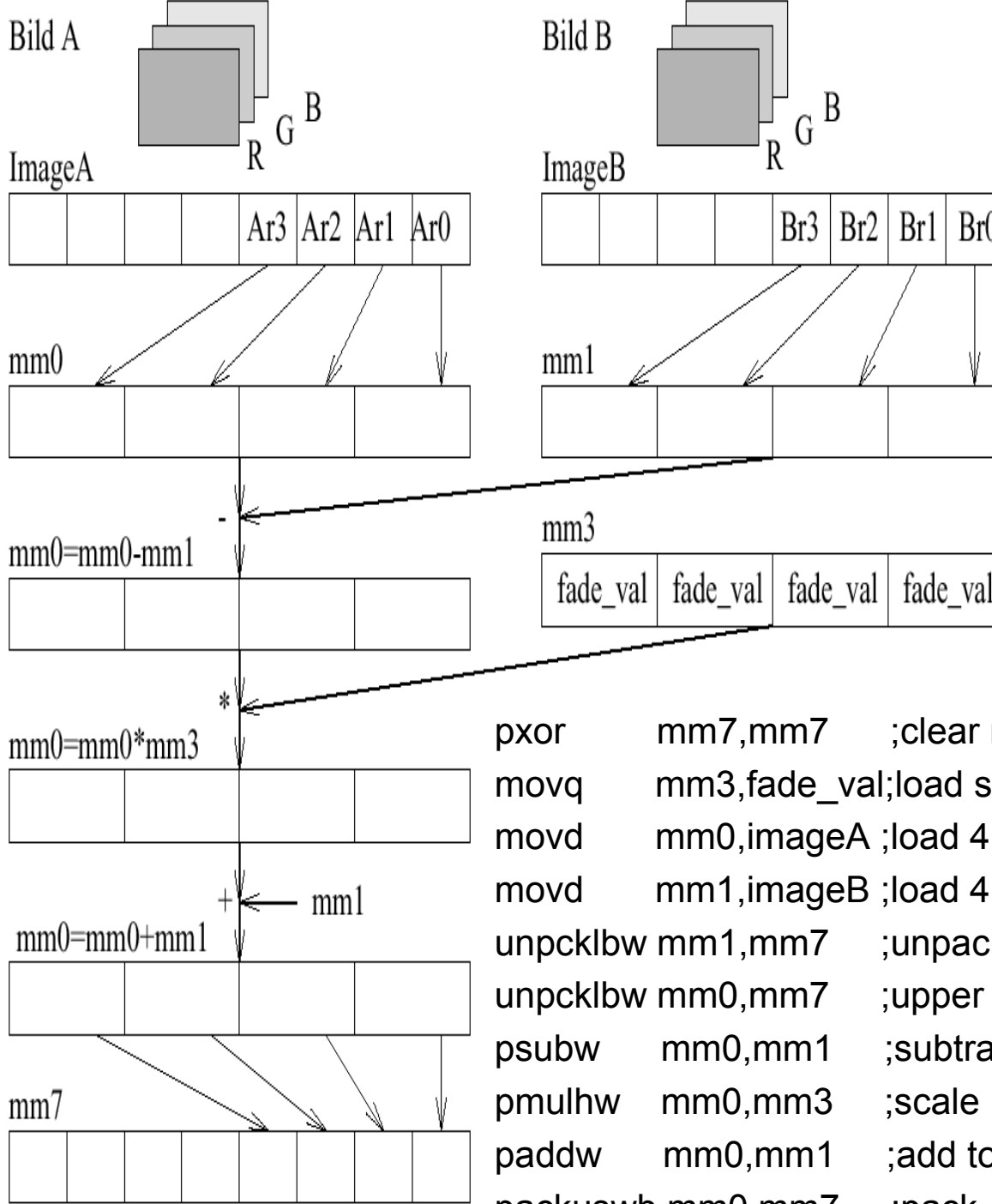
Psra[w/d] Psl[w/d/q] Psr[w/d/q]	No. of positions in register or instruction	Parallel shift of words, double words or 64 bit quad words
Punpckl[bw/wd/dq] Punpckh[bw/wd/dq]		Parallel unpack Parallel unpack
Packss[w/dw]	<i>saturating</i>	Parallel pack
Pand, Pandn Por, Pxor		Logical operations on 64 bit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

Application

Scaled interpolation between two images

Next word = next pixel, same color.

4 pixels processed at a time.



Summary

Hardware in a loop

- Sensors
- Discretization
- Information processing
 - Importance of energy efficiency
 - Special purpose HW very expensive
 - Energy efficiency of processors
 - Code size efficiency
 - Run-time efficiency
 - MPSoCs
 - Reconfigurable Hardware
- D/A converters
- Actuators