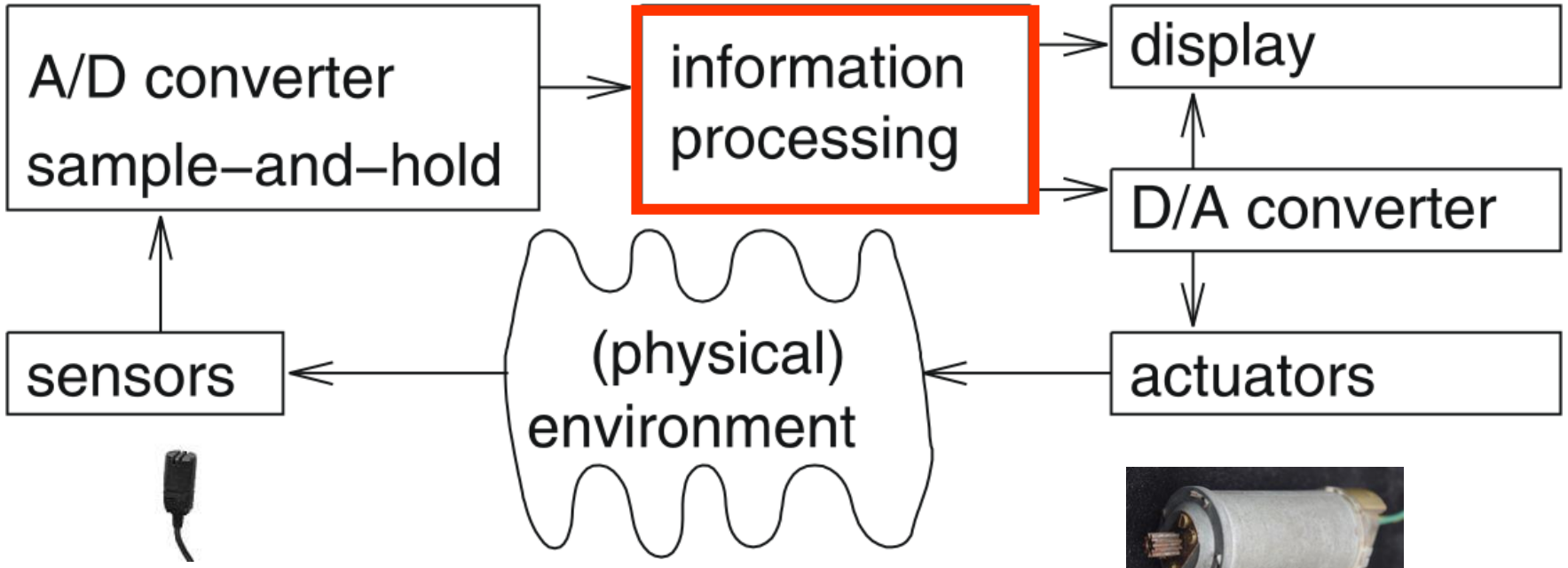
Embedded Systems 13

REVIEW: Embedded System Hardware

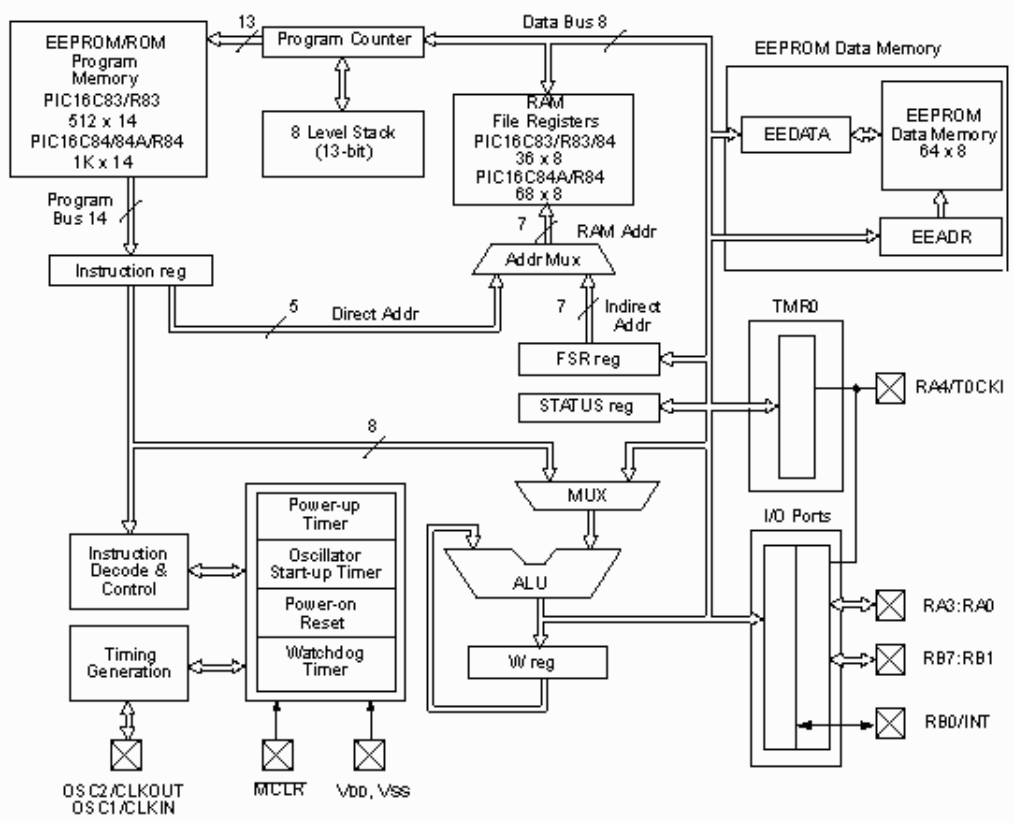
Embedded system hardware is frequently used in a loop (***"hardware in a loop"***):



☞ cyber-physical systems

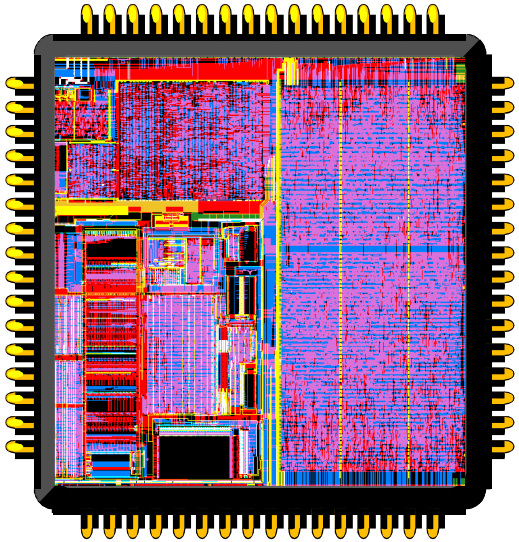
Microcontrollers

- Integrate several components of a microprocessor system onto one chip
  CPU, Memory, Timer, IO
- Low cost,
  small packaging
- Easy integration
  with circuits
- Single-purpose

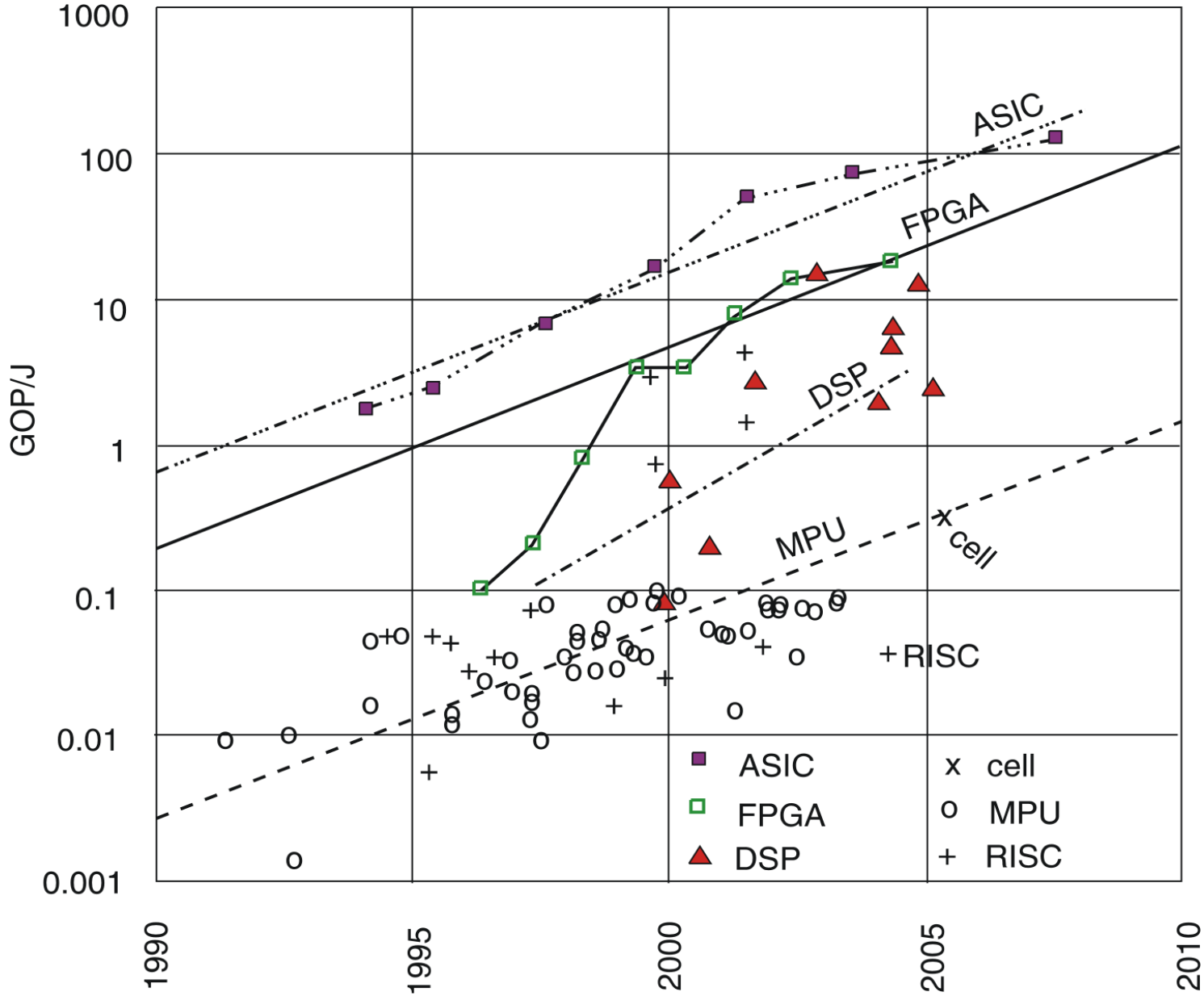


Application Specific Circuits (ASICS) or Full Custom Circuits

- Approach suffers from
  - long design times,
  - lack of flexibility (changing standards) and
  - high costs (e.g. Mill. $ mask costs).
- Custom-designed circuits necessary
  - if ultimate speed or
  - energy efficiency is the goal and
  - large numbers can be sold.



Energy



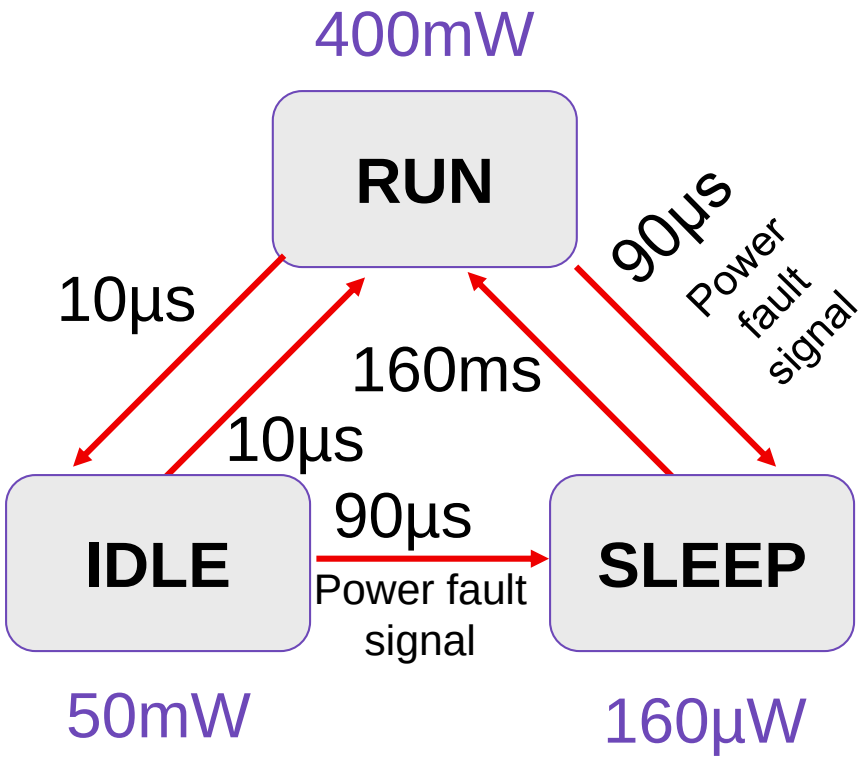
Low Power vs. Low Energy Consumption

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

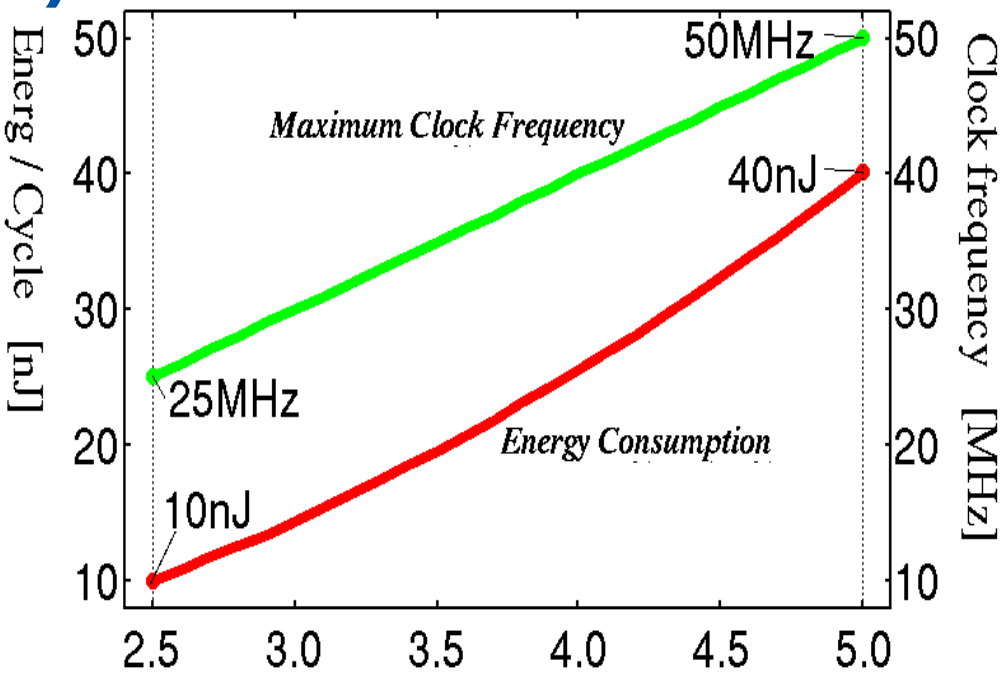
Dynamic power management (DPM)

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)



[Courtesy, Yasuura, 2000]

Power consumption of CMOS circuits (ignoring leakage):

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

α : switching activity

C_L : load capacitance

V_{dd} : supply voltage

f : clock frequency

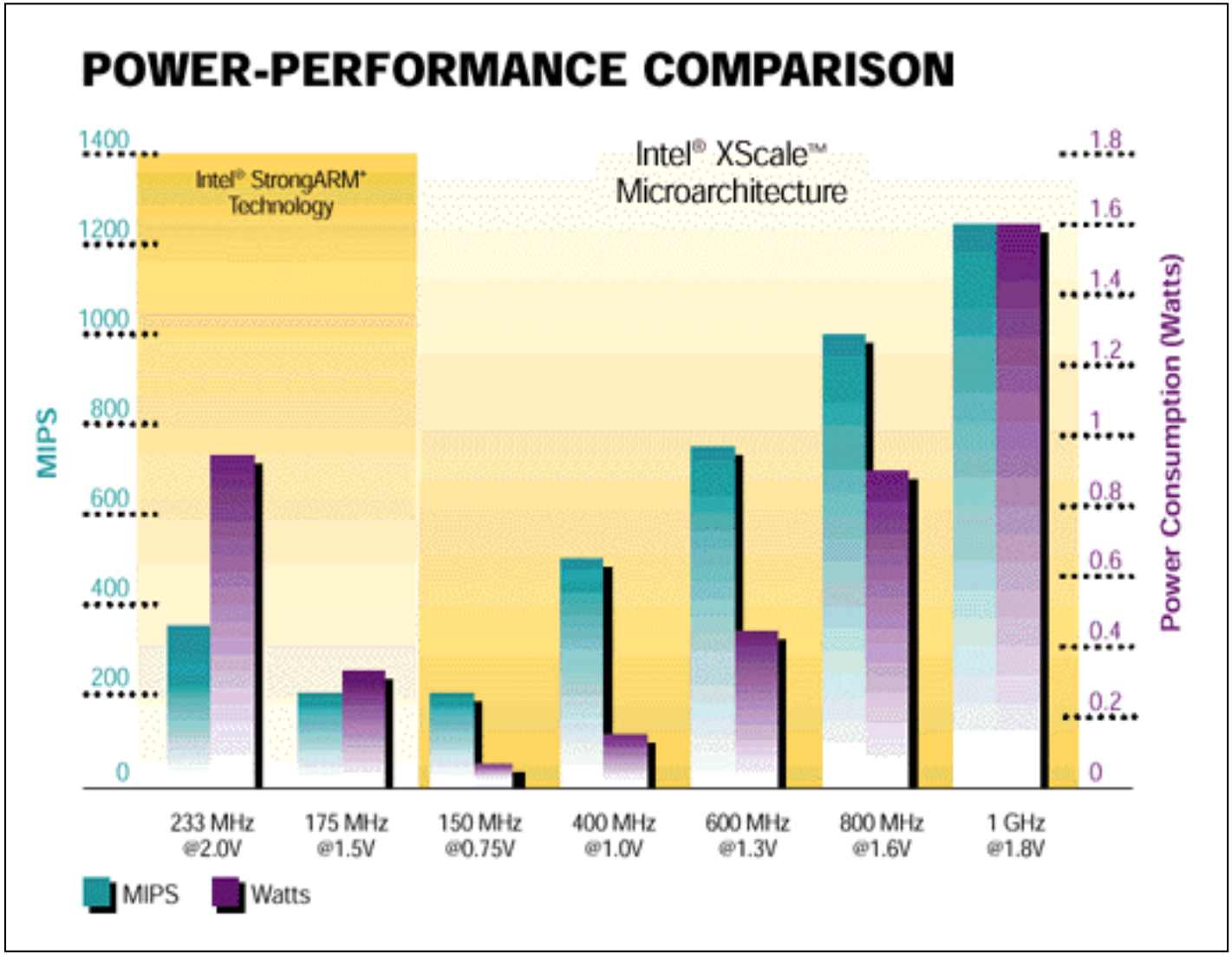
Delay for CMOS circuits:

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

V_t : threshhold voltage

(V_t < than V_{dd})

Variable-voltage/frequency example: INTEL Xscale



OS should schedule distribution of the energy budget.

From Intel's Web Site

Low voltage, parallel operation more efficient than high voltage, sequential 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
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:	$P° = P / \boldsymbol{\alpha^2}$ per operation,
Power for α operations per clock:	$P' = \boldsymbol{\alpha} \times P° = P / \boldsymbol{\alpha}$,
Time to run a program is still:	$t' = t$,
Energy required to run program:	$E' = P' \times t = E / \boldsymbol{\alpha}$

☞ Argument in favour of voltage scaling, VLIW processors, and multi-cores

Rough approxi-mations!

Application: VLIW processing and voltage scaling in the Crusoe processor

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

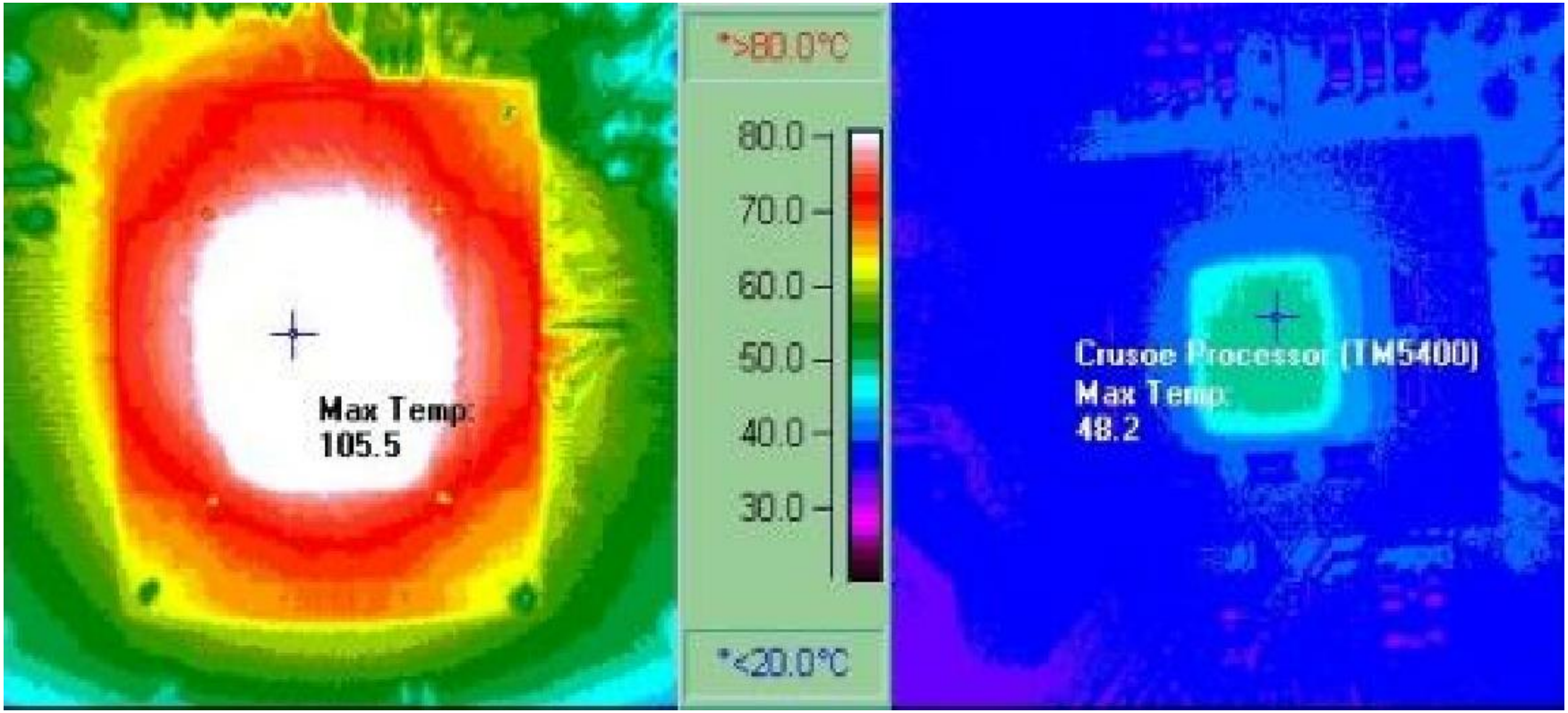
  Scaling is triggered when CPU load change is detected by software (~1/2 ms).
- More load: Increase of supply voltage (~20 ms/step), followed by scaling clock frequency
- Less load: reduction of clock frequency, followed by reduction of supply voltage

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

Result (as published by transmeta)

Pentium

Crusoe

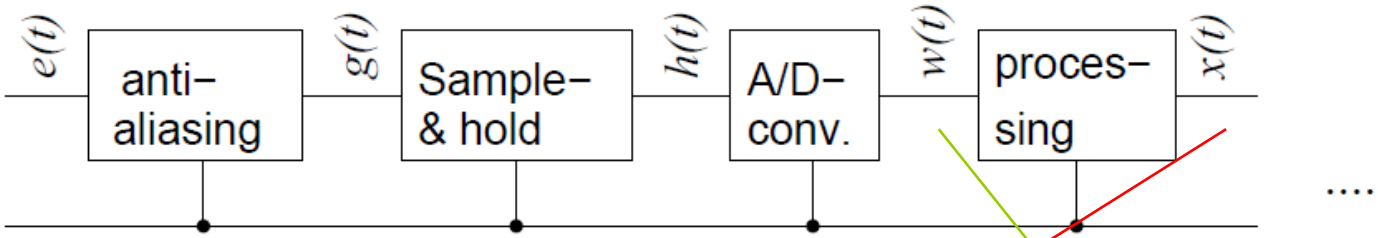


Running the same multimedia application.

[www.transmeta.com]

Digital Signal Processing (DSP)

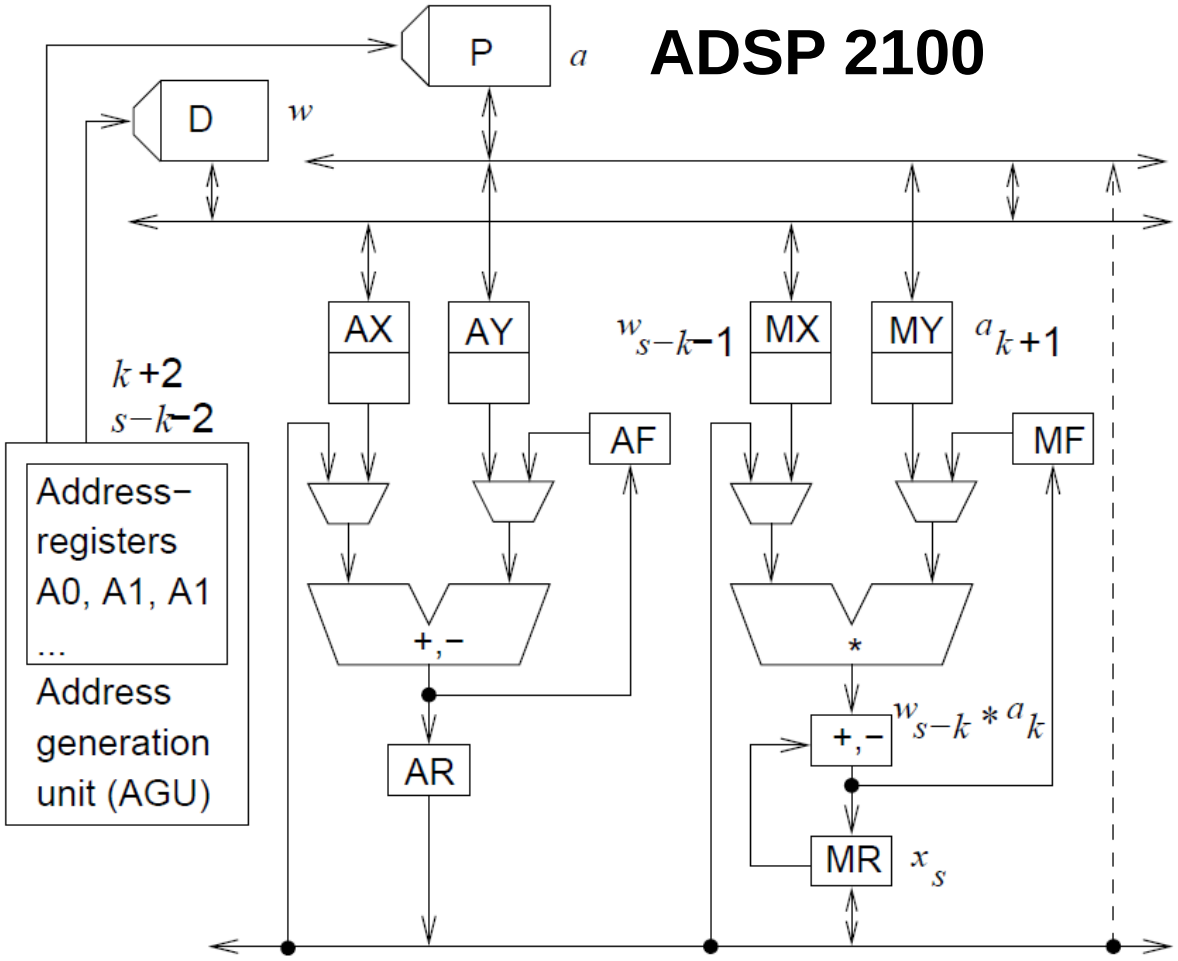
Example: Filtering



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

Signal at $t=t_s$ (sampling points)

Filtering in digital signal processing

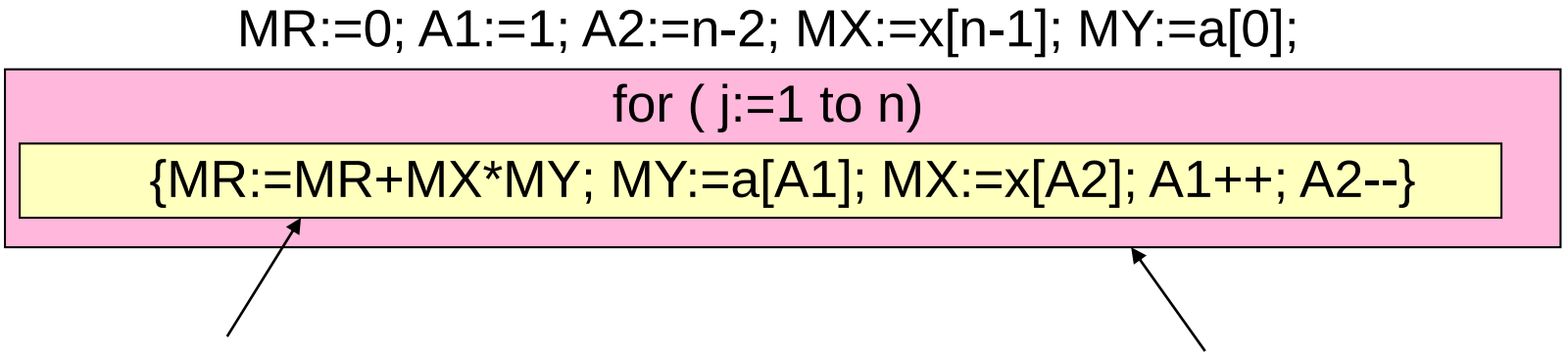


$$x_s = \sum_{k=0}^{n-1} w_{s-k} * a_k$$

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;
}
```

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

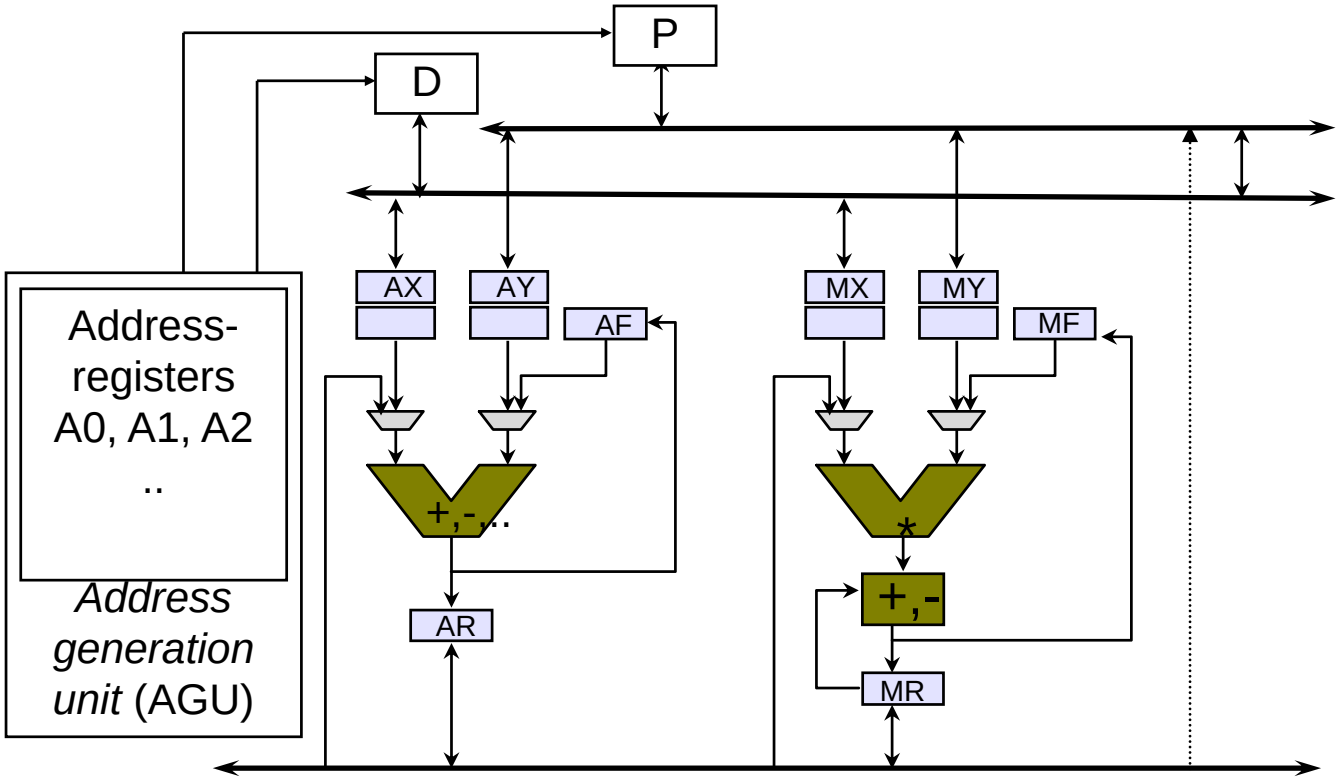


Multiply/accumulate (MAC) instruction

Zero-overhead loop (ZOL) instruction preceding MAC instruction.
Loop testing done in parallel to MAC operations.

Heterogeneous registers

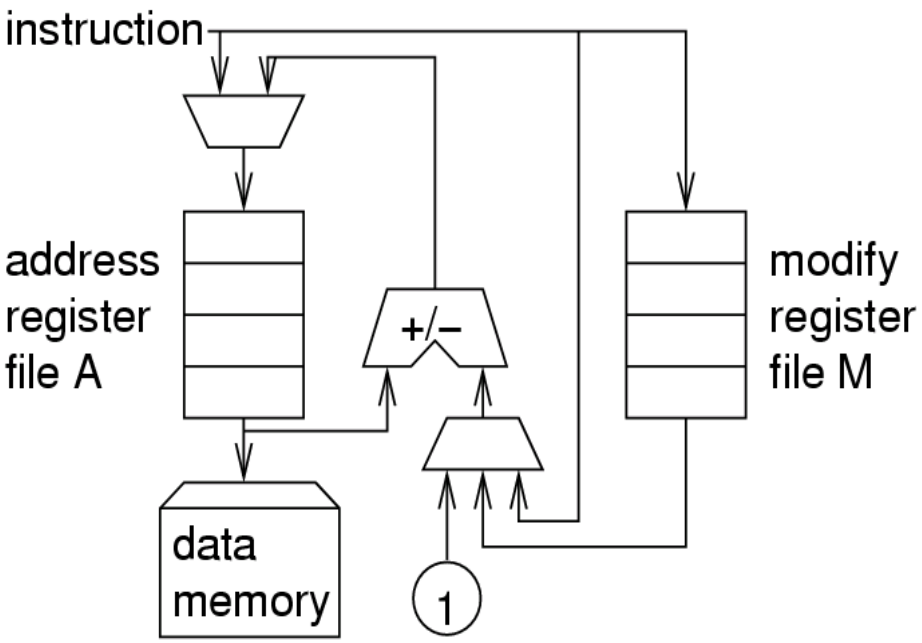
Example (ADSP 210x):



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

Separate address generation units (AGUs)

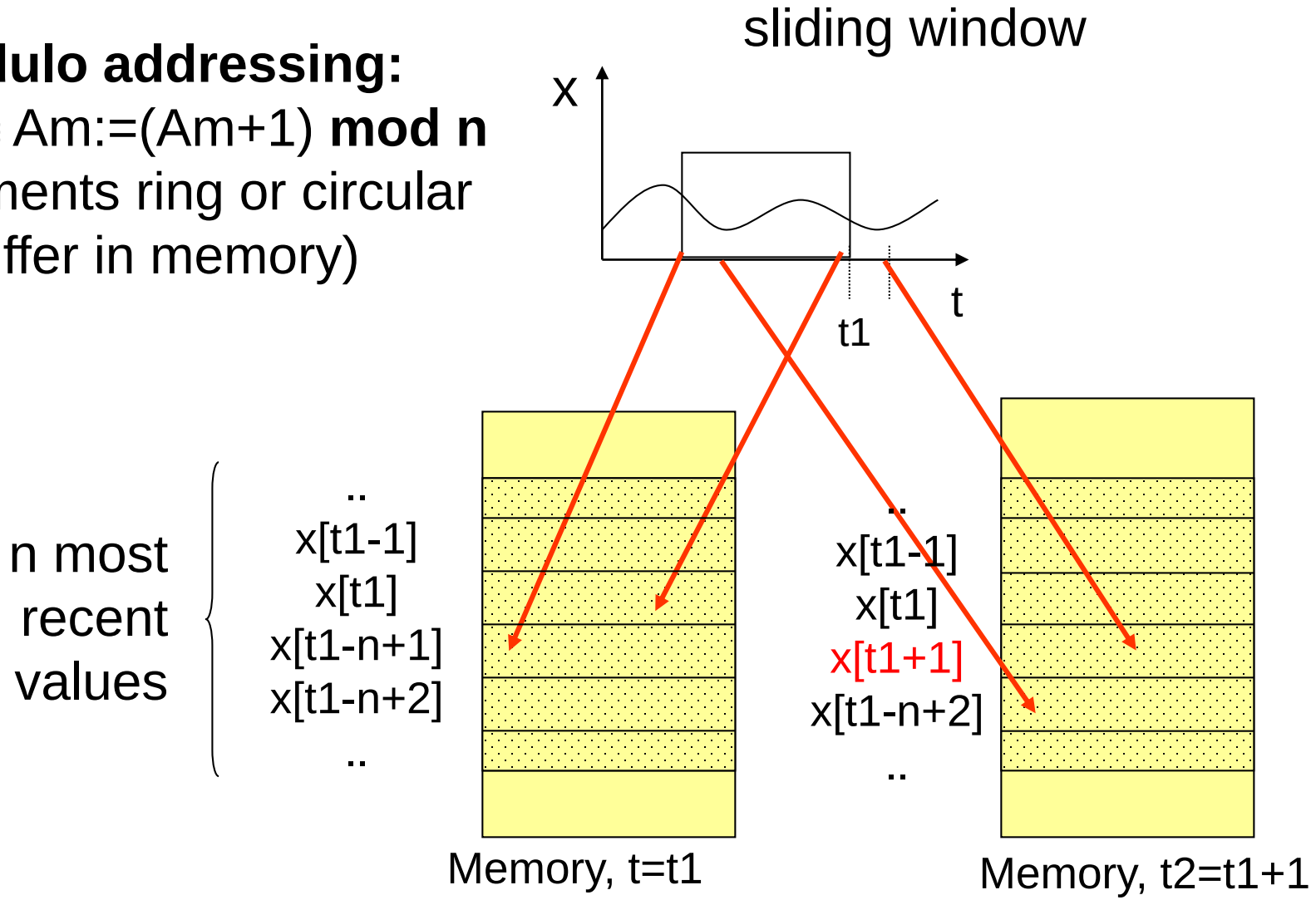
Example (ADSP 210x):



- 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 ± 1 also takes 0 time,
- same for A := A ± M;
- A := <immediate in instruction> requires extra instruction

Modulo addressing

Modulo addressing:
Am++ ≡ Am:=(Am+1) **mod n**
(implements ring or circular buffer in memory)



Saturating arithmetic

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

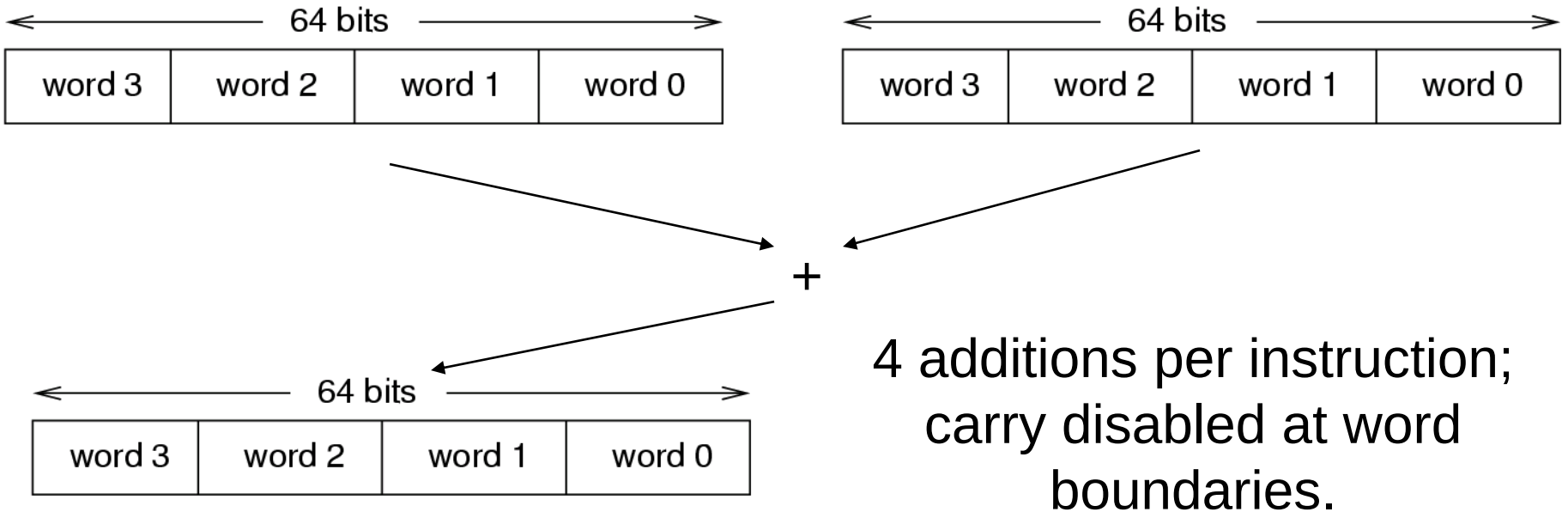
a		0111	
b	+	1001	
standard wrap around arithmetic		(1)0000	
saturating arithmetic		1111	
(a+b)/2: correct		1000	
wrap around arithmetic		0000	
saturating arithmetic + shifted		0111	„almost correct“

- 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.

Multimedia-Instructions/Processors

- Multimedia instructions exploit many registers, adders etc that 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)

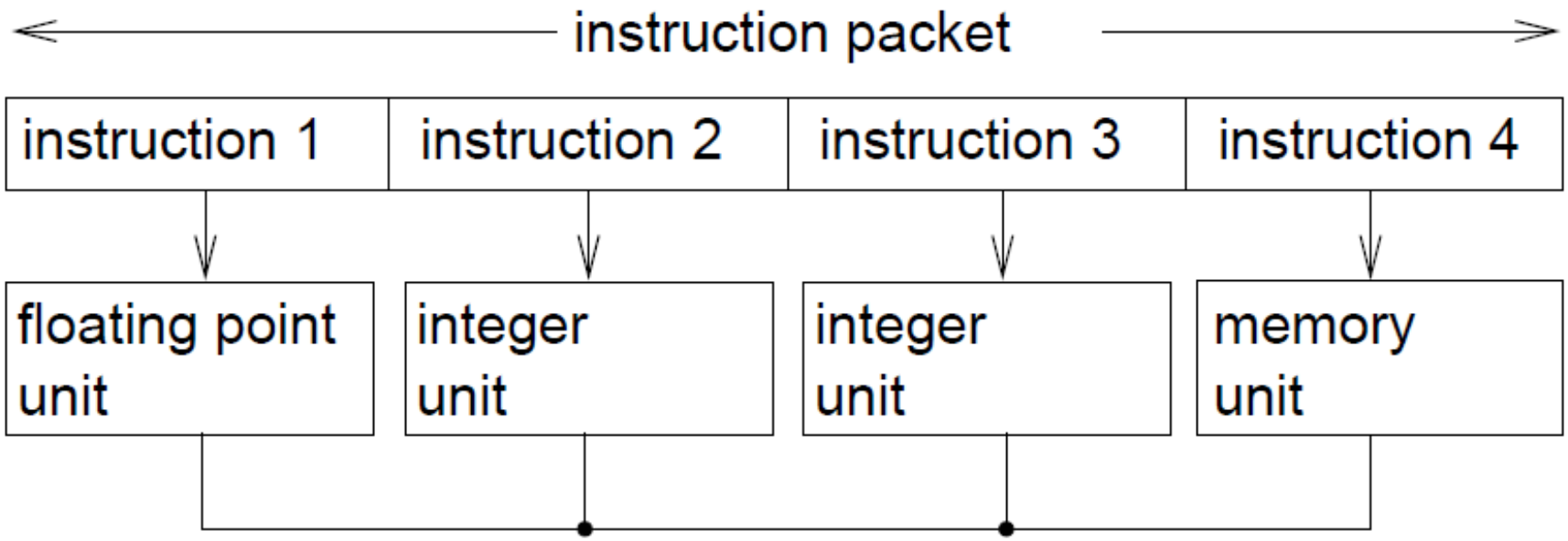
☞ 2-8 values can be stored per register and added. E.g.:



4 additions per instruction; carry disabled at word boundaries.

Key idea of very long instruction word (VLIW) computers

- Instructions included in long instruction packets. Instruction packets are assumed to be executed in parallel.
- Fixed association of packet bits with functional units.



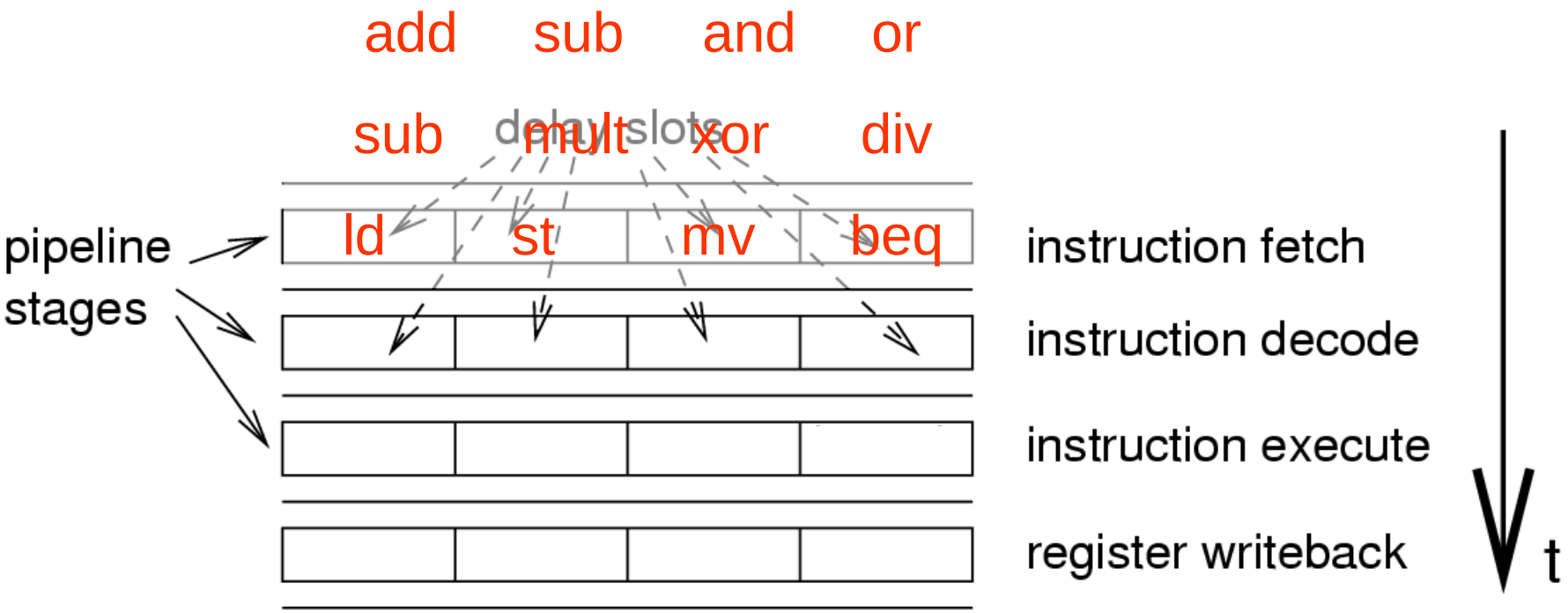
Very long instruction word (VLIW) architectures

- Very long instruction word
  ("instruction packet") contains several instructions, all of which are assumed to be executed in parallel.
- Compiler is assumed to generate these "parallel" packets
- Complexity of finding parallelism is moved from the hardware (RISC/CISC processors) to the compiler;
  Ideally, this avoids the overhead (silicon, energy, ..) of identifying parallelism at run-time.

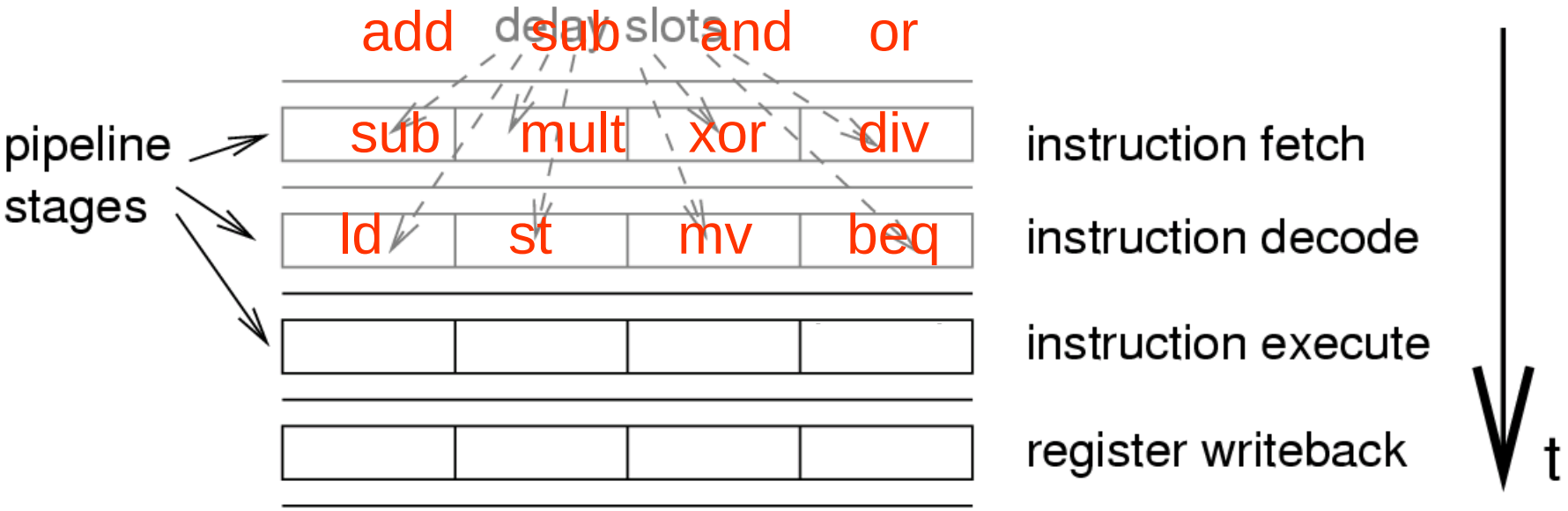
☞ A lot of expectations into VLIW machines

- Explicitly parallel instruction set computers (EPICs) are an extension of VLIW architectures: parallelism detected by compiler, but no need to encode parallelism in 1 word.

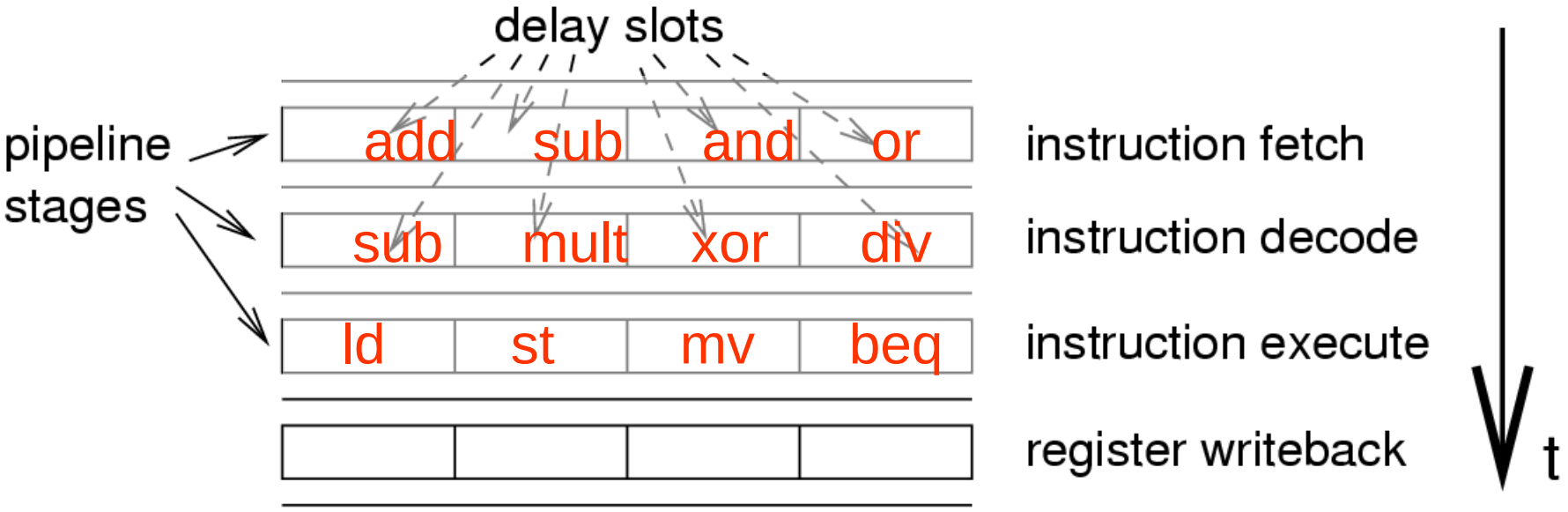
Large # of delay slots, a problem of VLIW processors



Large # of delay slots, a problem of VLIW processors



Large # of delay slots, a problem of VLIW processors



The execution of many instructions has been started before it is realized that a branch was required.
Nullifying those instructions would waste compute power

☞ Executing those instructions is declared a feature, not a bug.

☞ How to fill all “delay slots“ with useful instructions?

☞ Avoid branches wherever possible.

Predicated execution: Implementing IF-statements „branch-free“

Conditional Instruction „[c] I“ consists of:

- **condition c**
- **instruction I**

c = true => I executed
c = false => NOP

Predicated execution: Implementing IF-statements „branch-free“: TI C6x

```
if (c)
{ a = x + y;
  b = x + z;
}
else
{ a = x - y;
  b = x - z;
}
```

Conditional branch

```
      [c] B L1
          NOP 5
          B L2
          NOP 4
          SUB x,y,a
      ||  SUB x,z,b
L1:       ADD x,y,a
      ||  ADD x,z,b
L2:
```

max. 12 cycles

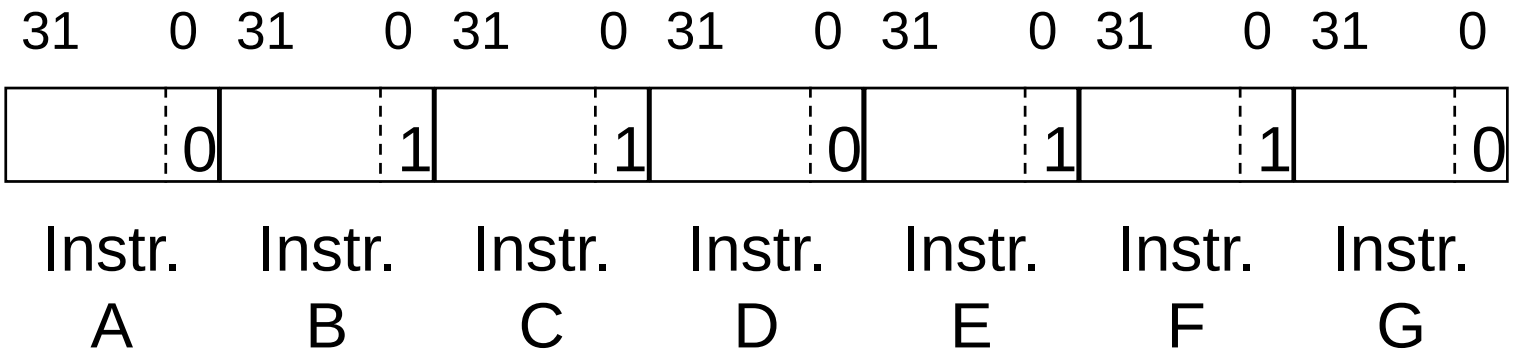
Predicated execution

```
     [c]  ADD x,y,a
|| [c]  ADD x,z,b
|| [!c] SUB  x,y,a
|| [!c] SUB  x,z,b
```

1 cycle

EPIC: TMS 320C6xx as an example

1 Bit per instruction encodes end of parallel exec.



Cycle	Instruction		
1	A		
2	B	C	D
3	E	F	G

Instructions B, C and D use disjoint functional units, cross paths and other data path resources. The same is also true for E, F and G.

Parallel execution cannot span several packets.