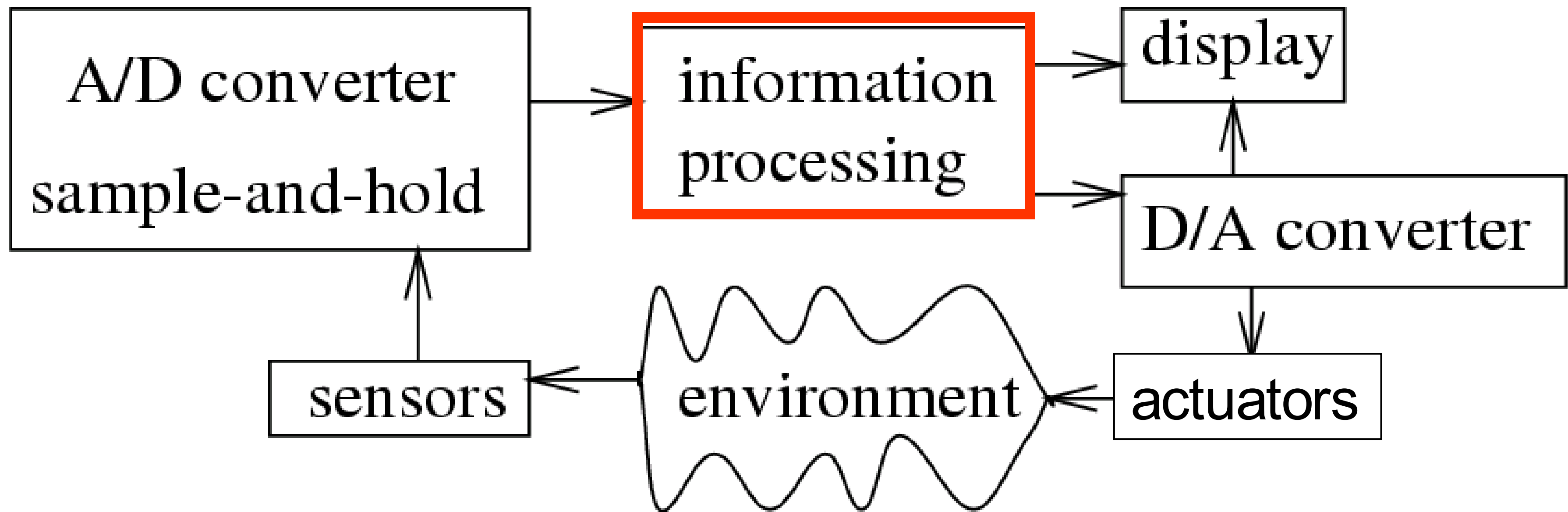
Embedded System Hardware
- Processing -

Peter Marwedel
Informatik 12
TU Dortmund
Germany

2008/11/18

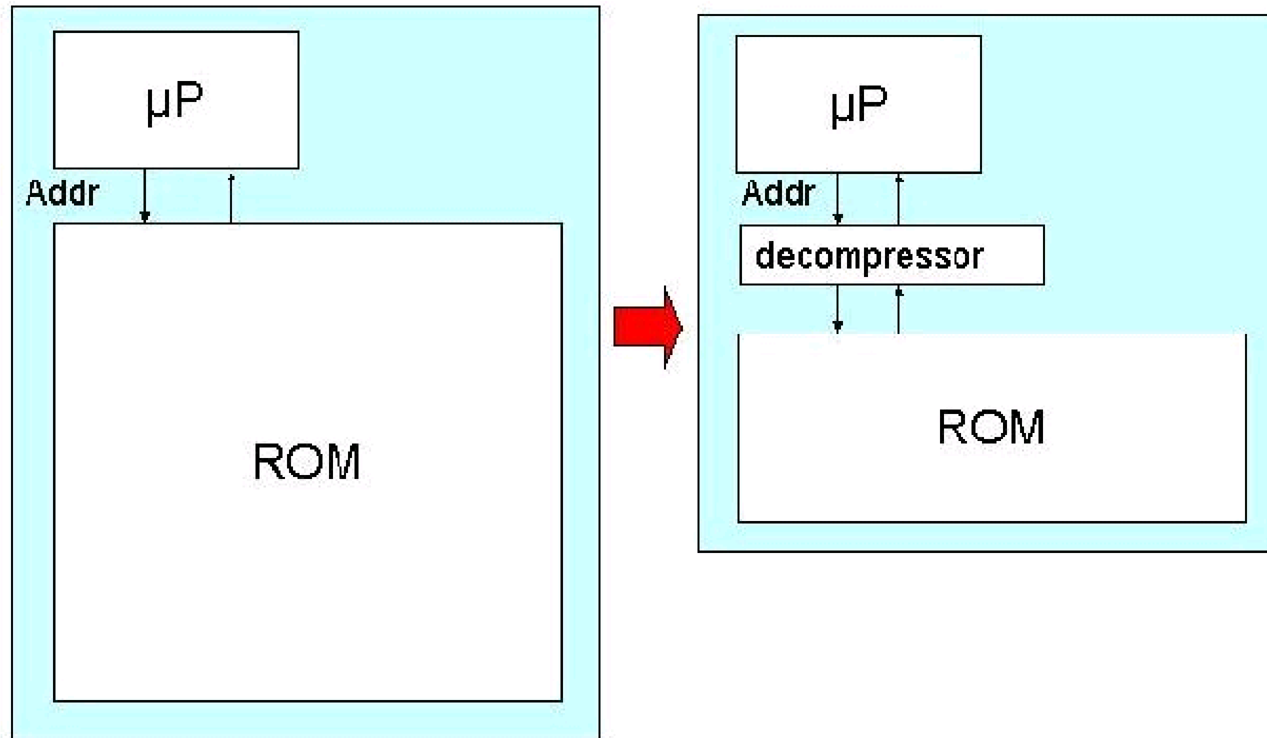
Embedded System Hardware

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



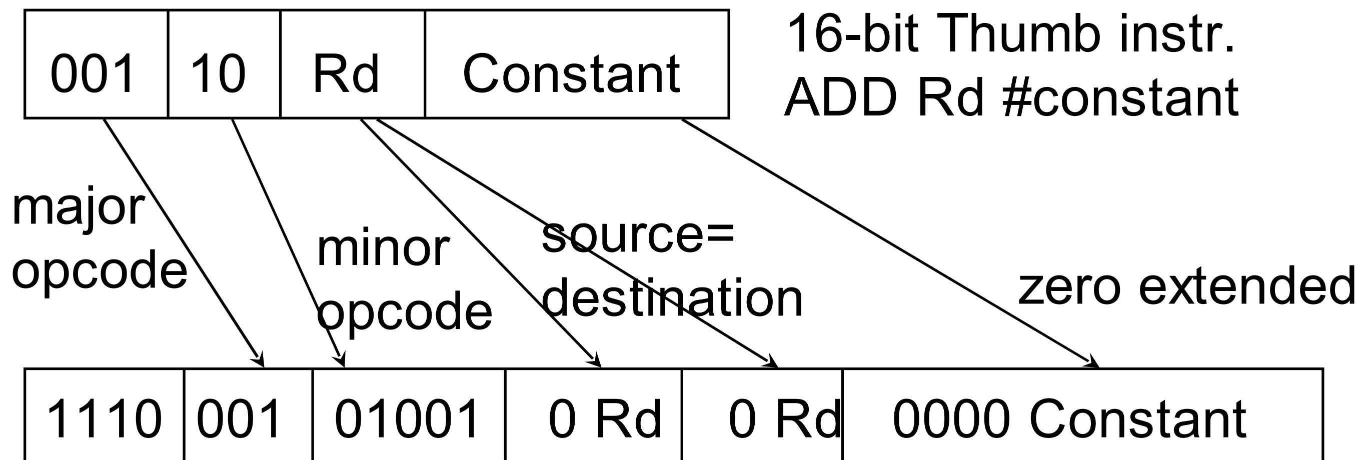
Key requirement #2: Code-size efficiency

- **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]

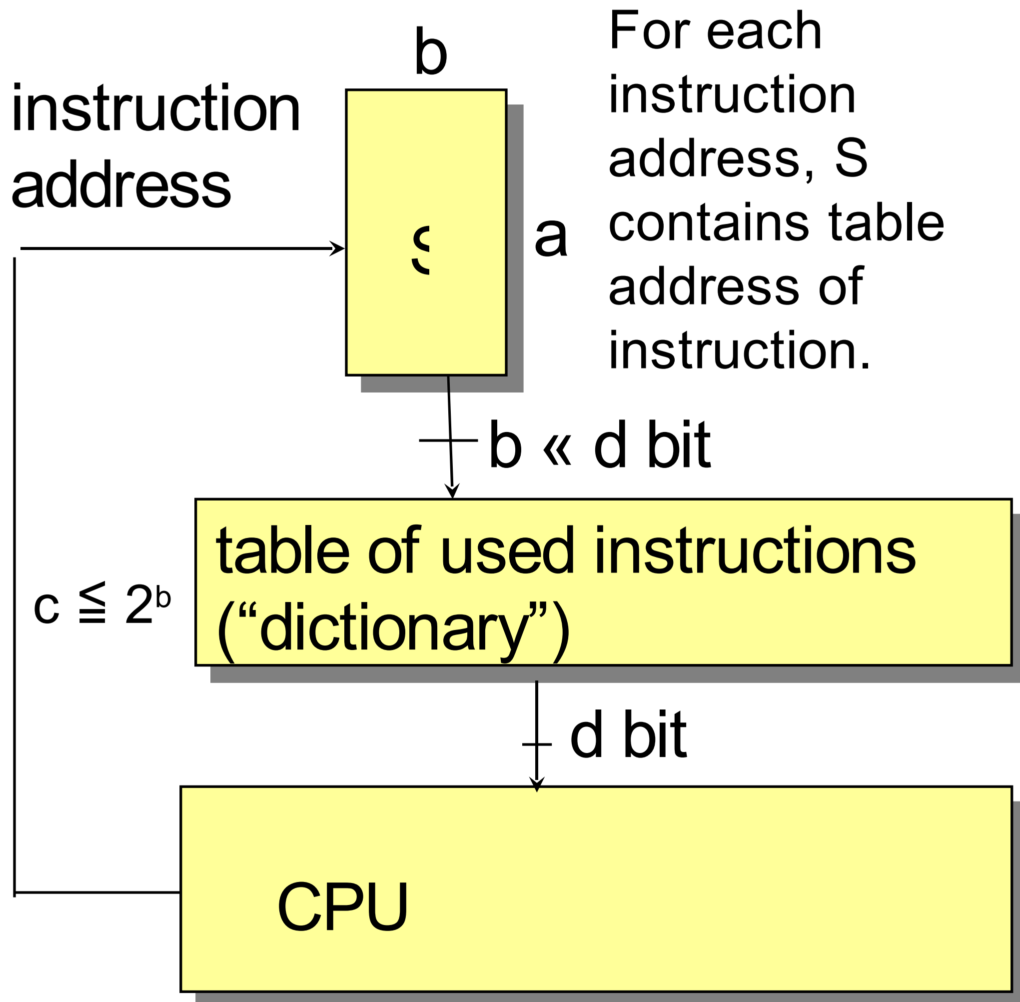
Same approach for LSI TinyRisc, …
Requires support by compiler, assembler etc.

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. Beszédes 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.

small

Hopefully, $a \times b + c \times d < a \times d$.

Called nanoprogramming in the Motorola 68000.

Cache-based decompression

- Main idea: decompression whenever cache-lines are fetched 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 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.iro.umontreal.ca/~latendre/compactBib/

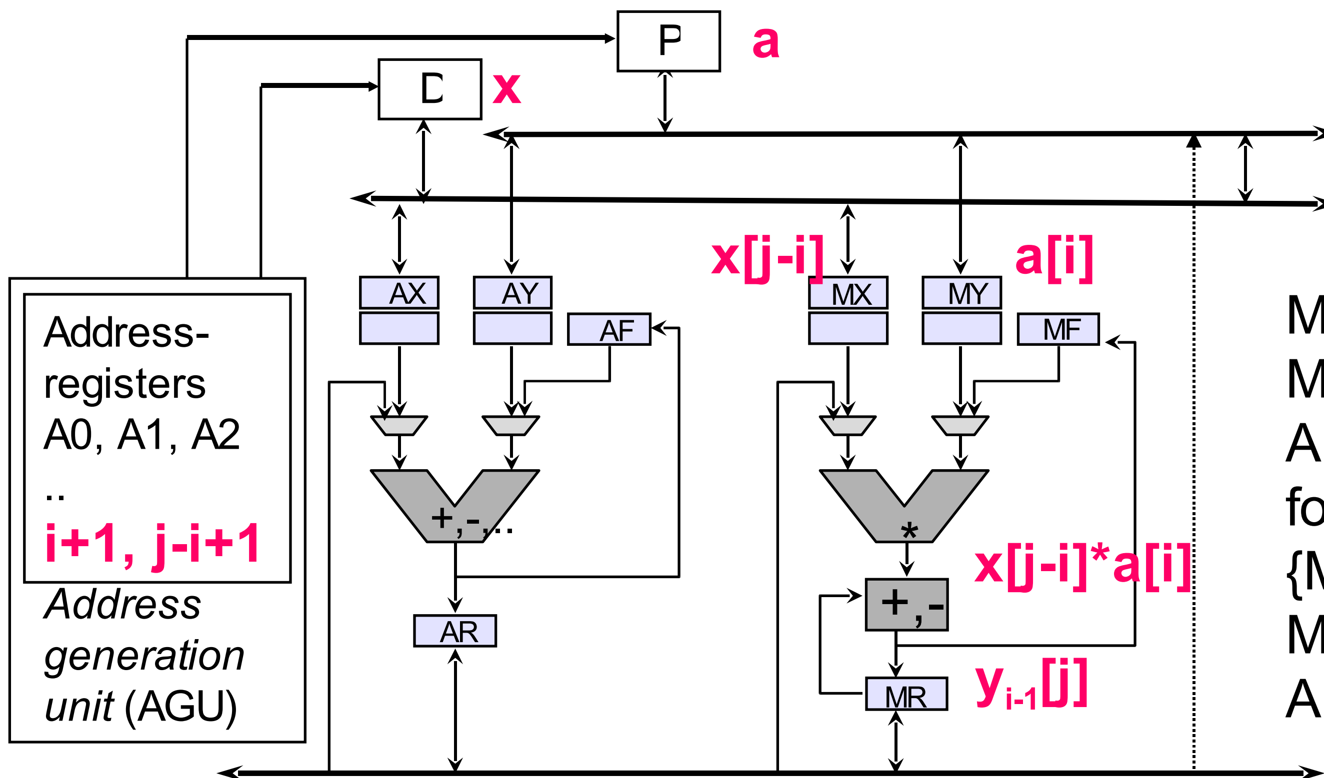
  (153 entries as per 11/2004)

Key requirement #3: Run-time efficiency
- Domain-oriented architectures -

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

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

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**

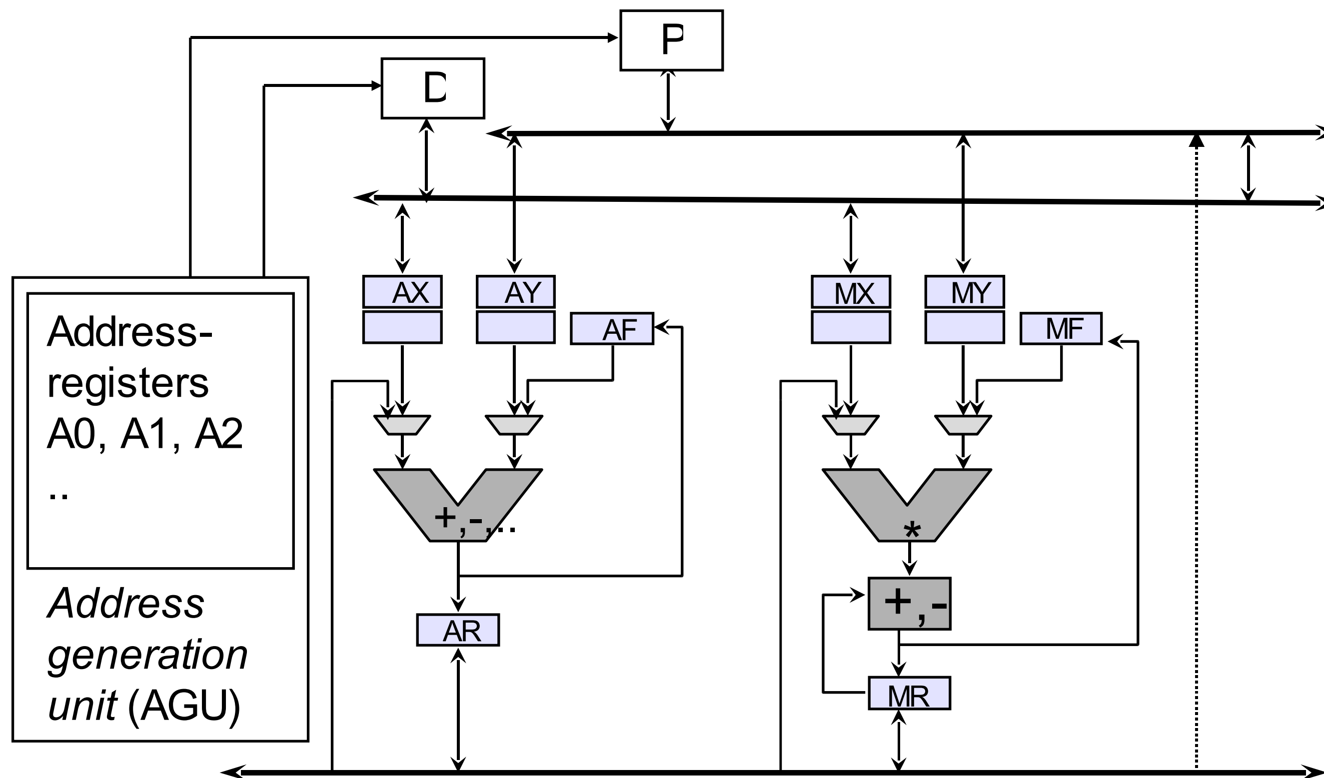
```
MR:=0; A1:=1; A2:=n-2; MX:=x[n-1]; MY:=a[0];
for ( j:=1 to n)
  {MR:=MR+MX*MY; MY:=a[A1]; MX:=x[A2]; A1++; A2--}
```

Multiply/accumulate (MAC) instruction

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

Heterogeneous registers

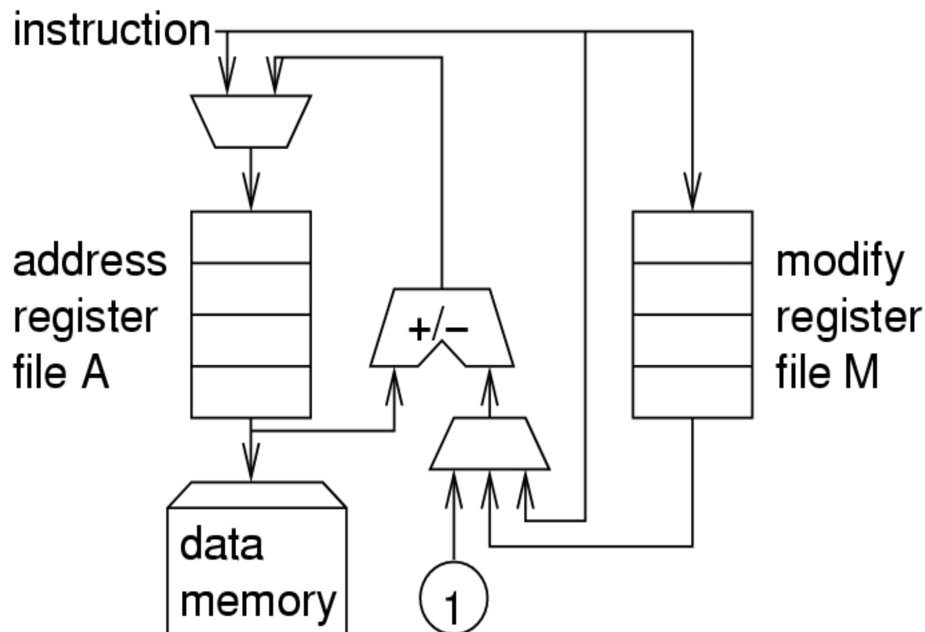
Example (ADSP 210x):



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

Separate address generation units (AGUs)

Example (ADSP 210x):



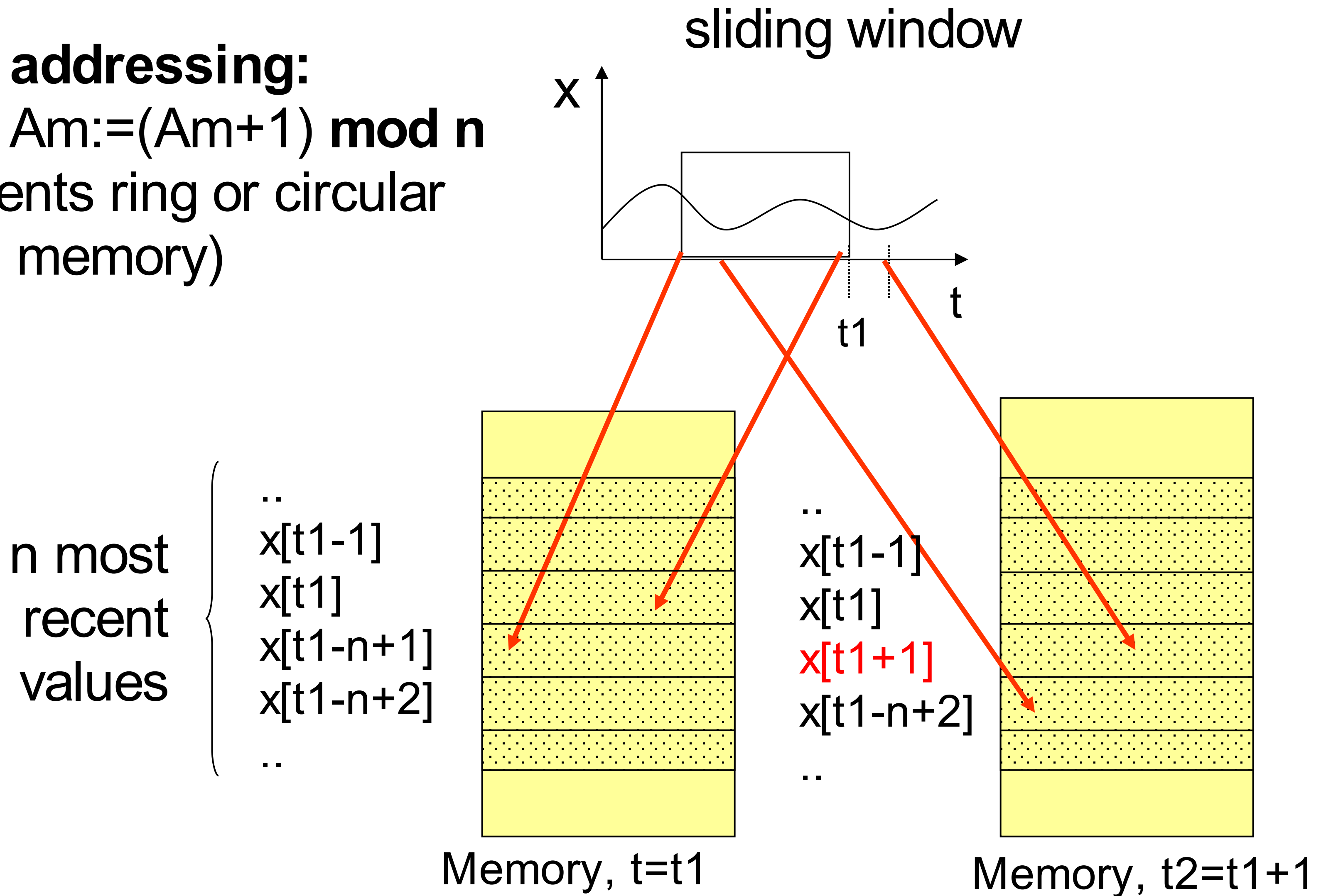
- Data memory can only be fetched with address contained in A,
- but this can be done in parallel with operation in main data path (takes effectively 0 time).
- A := A ± 1 also takes 0 time,
- same for A := A ± M;
- A := <immediate in instruction> requires extra instruction

☞ Minimize load immediates

☞ Optimization in optimization chapter

Modulo addressing

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



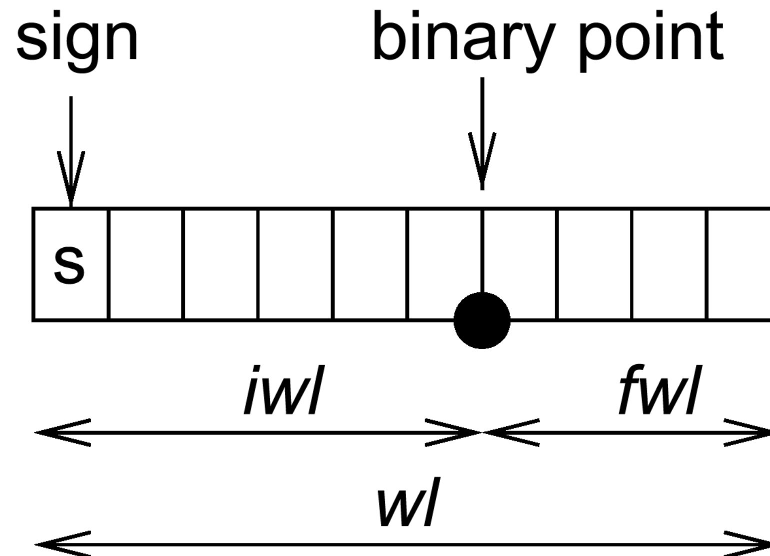
Saturating arithmetic

- 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.

Fixed-point arithmetic



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

Properties of fixed-point arithmetic

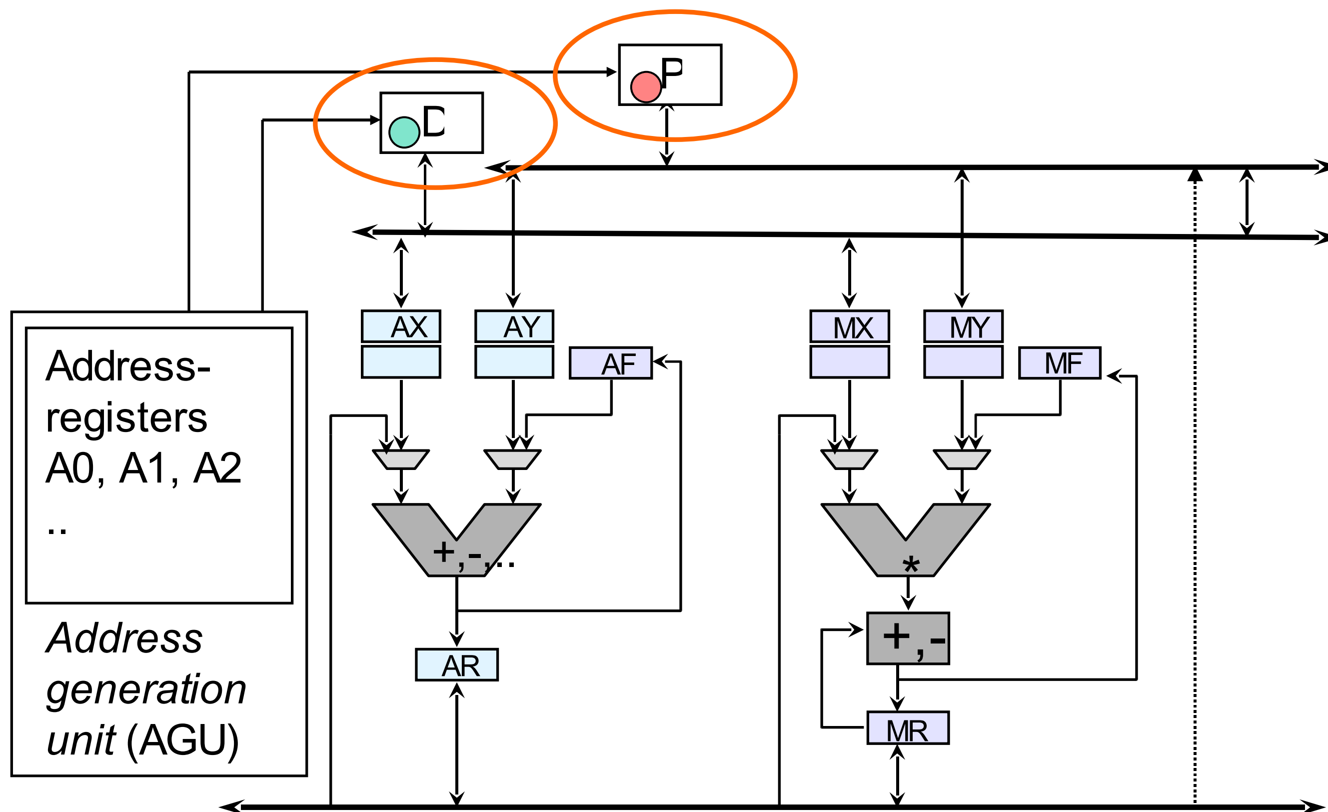
- Automatic scaling a key advantage for multiplications.
- Example:
  x= 0.5 x 0.125 + 0.25 x 0.125 = 0.0625 + 0.03125 = 0.09375
  For *iwl*=1 and *fwl*=3 decimal digits, the less significant digits are automatically chopped off: x = 0.093
  Like a floating point system with numbers $\in$ (-1..1),
  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

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

Multiple memory banks or memories

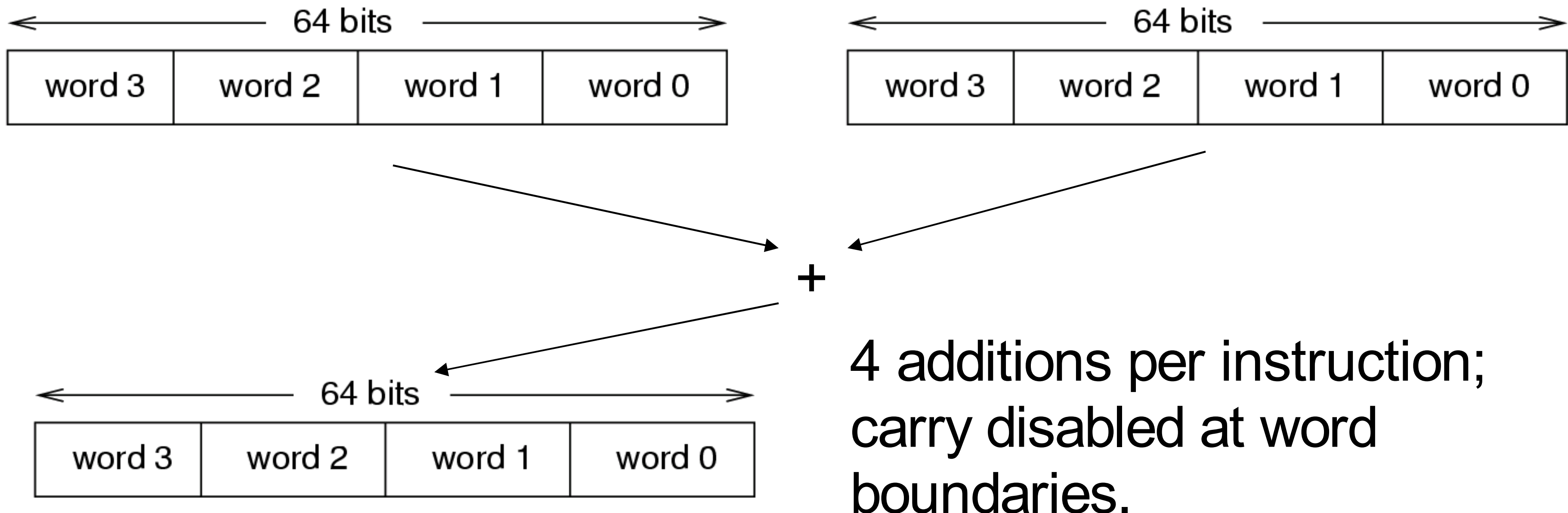


Simplifies parallel fetches

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.:

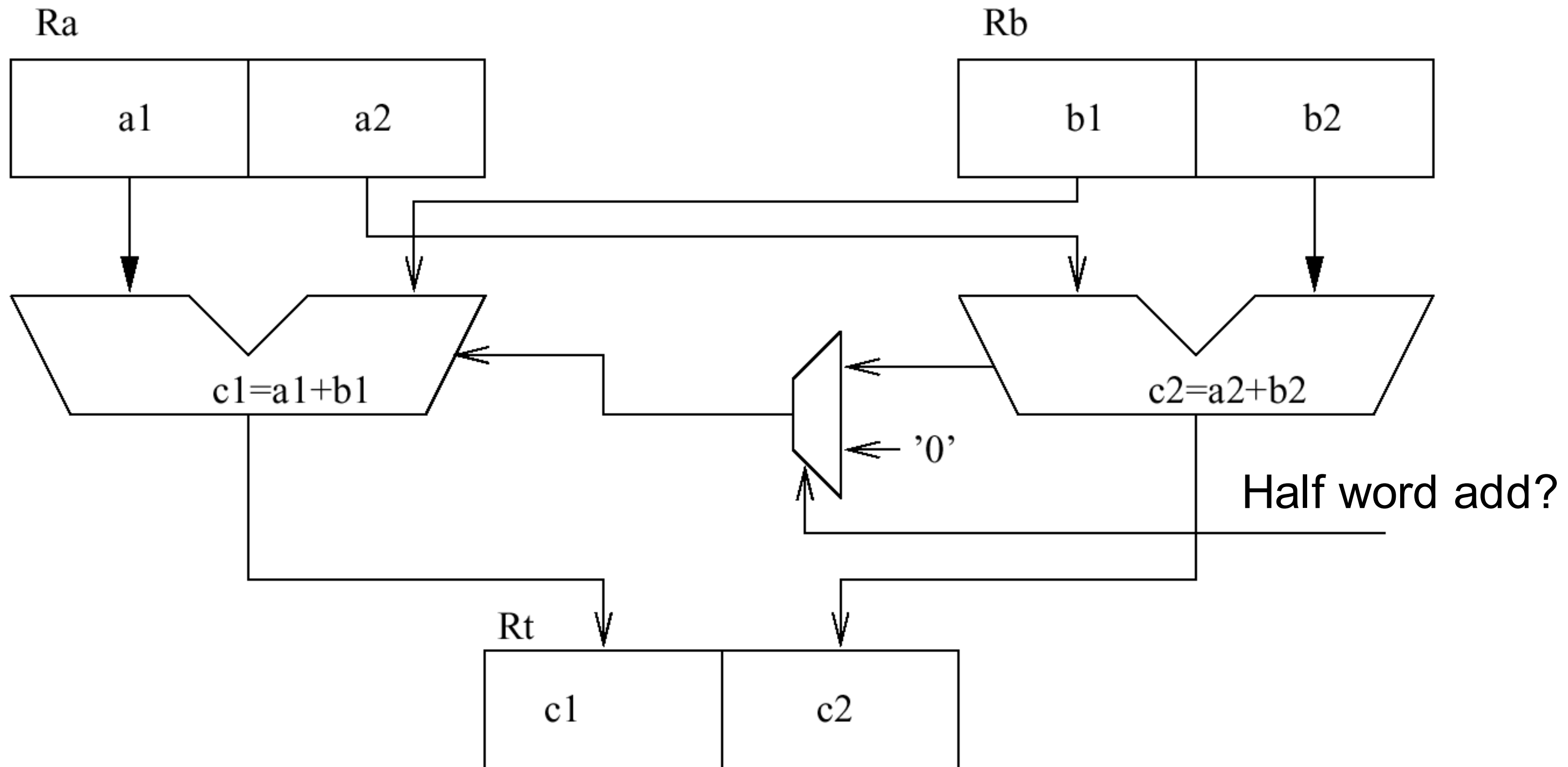


4 additions per instruction; carry disabled at word boundaries.

© p.marwedel, informatik 12, 2008

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]	*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)

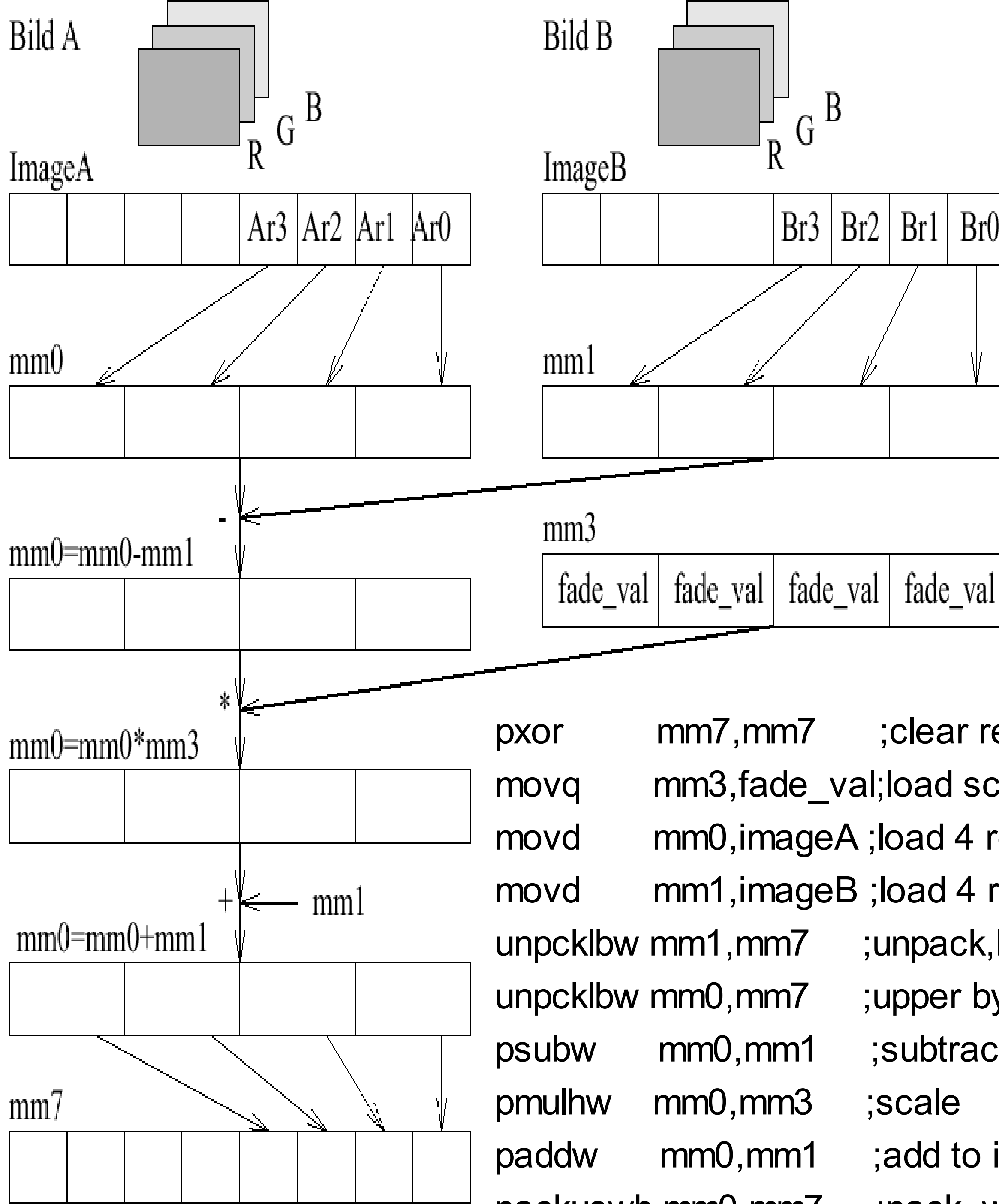
Psra[w/d] Psll[w/d/q] Psrl[w/d/q]	No. of positions in register or instruction	Parallel shift of words, double words or 64 bit quad words
Punpckl[bw/wd/dq] Punpckh[bw/wd/dq]		Parallel unpack Parallel unpack
Packss[wb/dw]	*saturating*	Parallel pack
Pand, Pandn Por, Pxor		Logical operations on 64 bit words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

Appli-cation

Scaled interpolation between 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
```

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 general 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.

EPIC: TMS 320C6xx as an example

Bit in each instruction encodes end of parallel execution

31 0 31 0 31 0 31 0 31 0 31 0 31 0

Instr. A | Instr. B | Instr. C | Instr. D | Instr. E | Instr. F | Instr. G

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

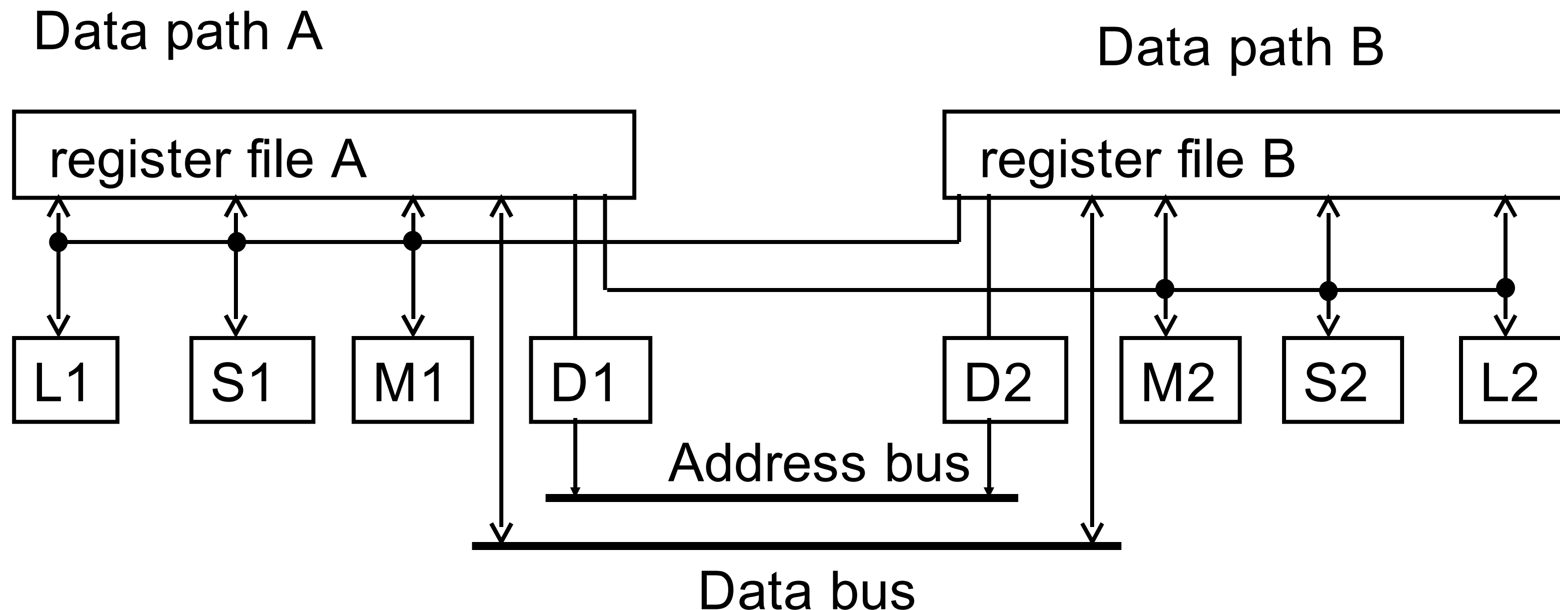
Instructions B, C and D use disjoint functional units, cross paths and other data path resources. The same is also true for E, F and G.

Parallel execution cannot span several packets.

Partitioned register files

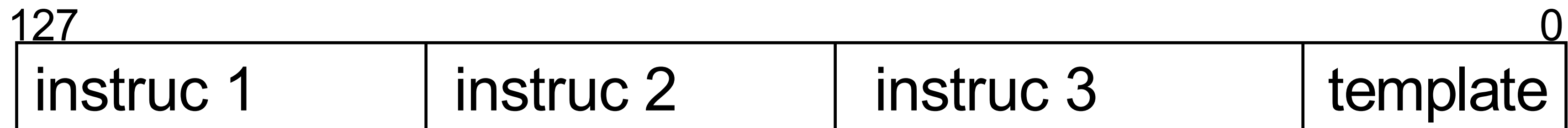
- Many memory ports are required to supply enough operands per cycle.
- Memories with many ports are expensive.

☞ Registers are partitioned into (typically 2) sets, e.g. for TI C60x:



More encoding flexibility with IA-64 Itanium

3 instructions per **bundle:**



There are 5 instruction types:

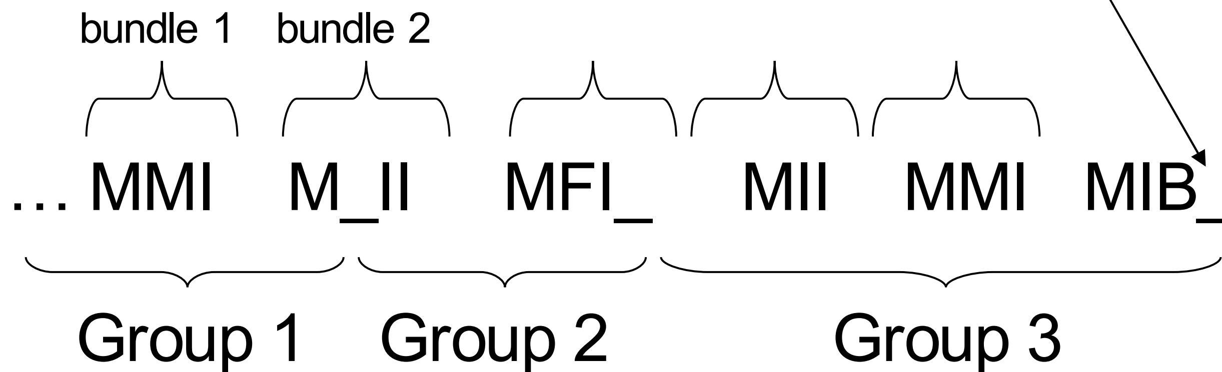
- A: common ALU instructions
- I: more special integer instructions (e.g. shifts)
- M: Memory instructions
- F: floating point instructions
- B: branches

The following combinations can be encoded in templates:

- MII, MMI, MFI, MIB, MMB, MFB, MMF, MBB, BBB, MLX with LX = *move 64-bit immediate* encoded in 2 slots

Templates and instruction types

End of parallel execution called **stops.**
Stops are denoted by underscores.
Example:



Very restricted placement of stops within bundle.
Parallel execution within groups possible.
Parallel execution can span several bundles

Instruction types are mapped to functional unit types

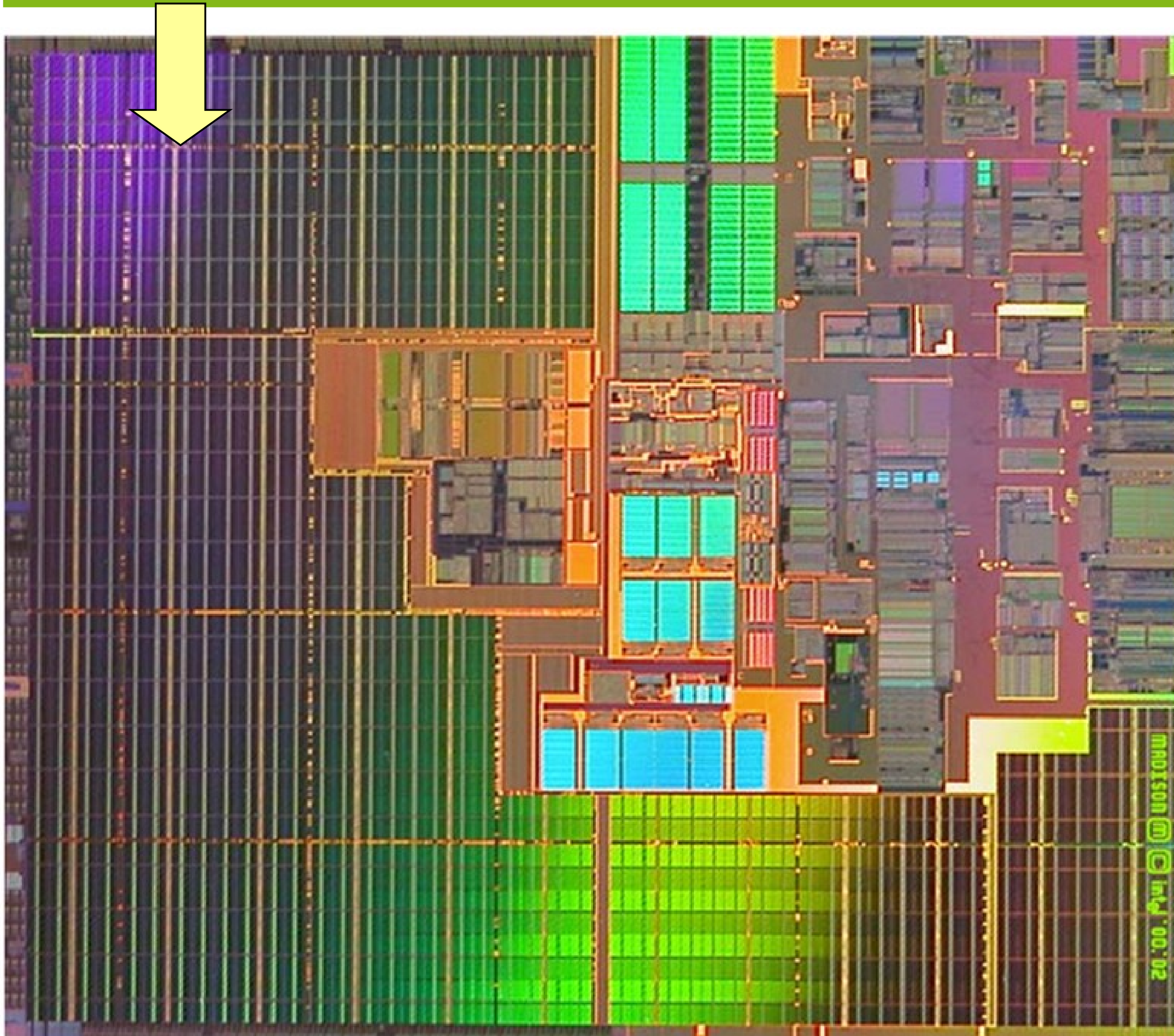
There are 4 functional unit (FU) types:

- M: Memory Unit
- I: Integer Unit
- F: Floating-Point Unit
- B: Branch Unit

Instruction types $\rightarrow$ corresponding FU type,
except type A (mapping to either I or M-functional units).

Implementation: Itanium 2 (2003)

L3 cache



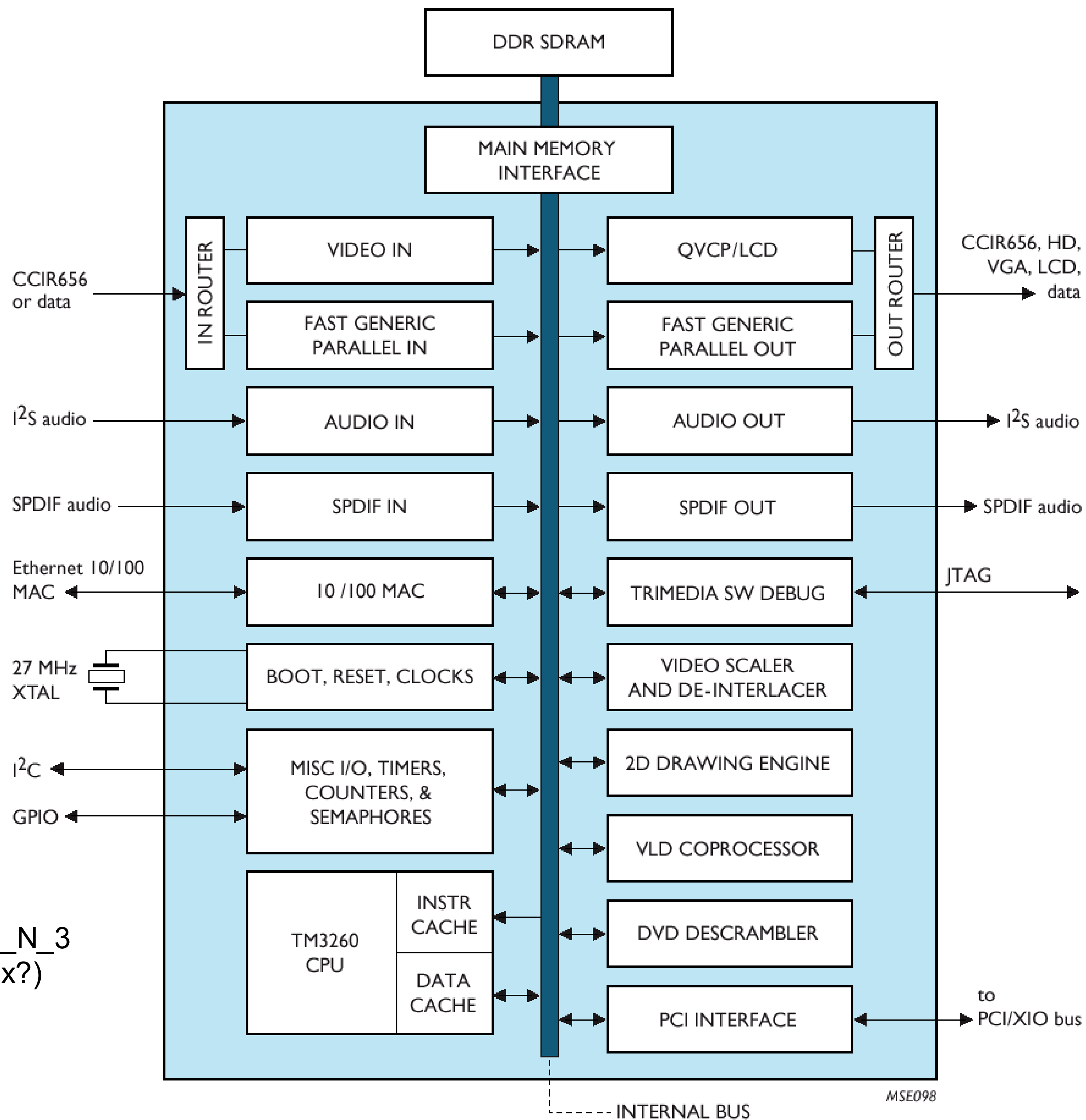
- 410M transistors
- 374 mm² die size
- 6MB on-die L3 cache
- 1.5 GHz at 1.3V

[ftp://download.intel.com/design/itanium2/download/madison_slides_r1.pdf]

© Intel, 2003

Philips TriMedia-Processor

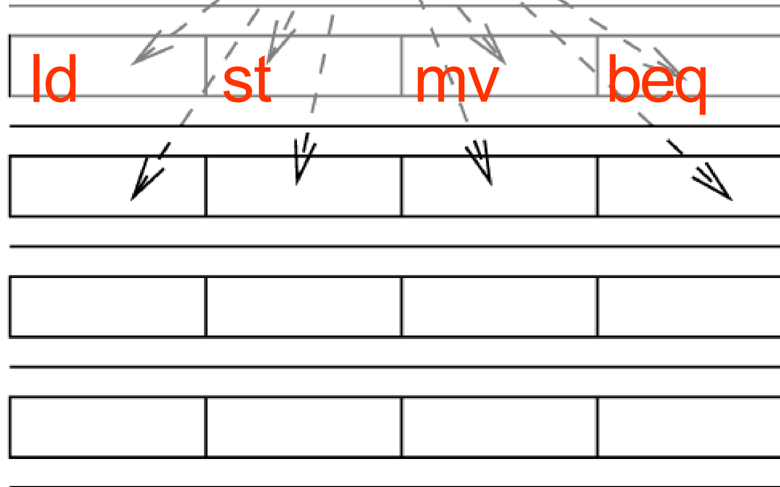
For multimedia-applications, up to 5 instructions/ cycle.



http://www.nxp.com/acrobat/datasheets/PNX15XX_SER_N_3.pdf (incompatible with firefox?)

© NXP

Large # of delay slots, a problem of VLIW processors

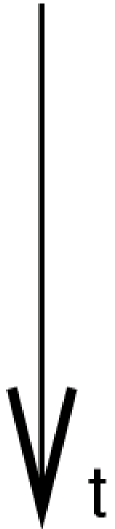


instruction fetch

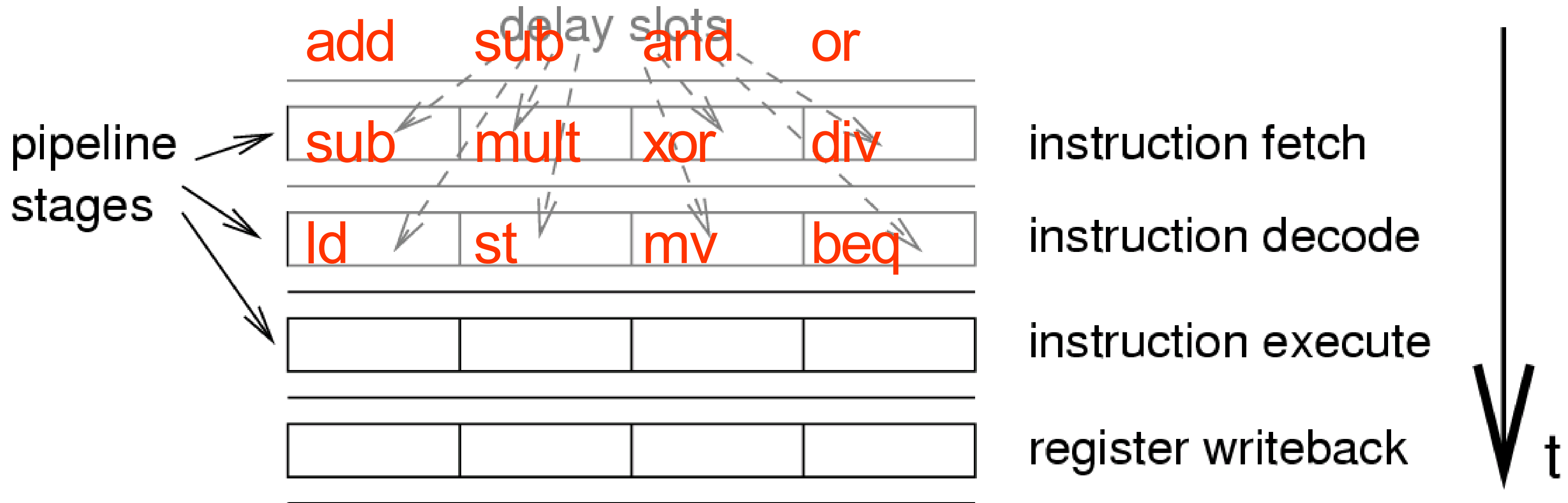
instruction decode

instruction execute

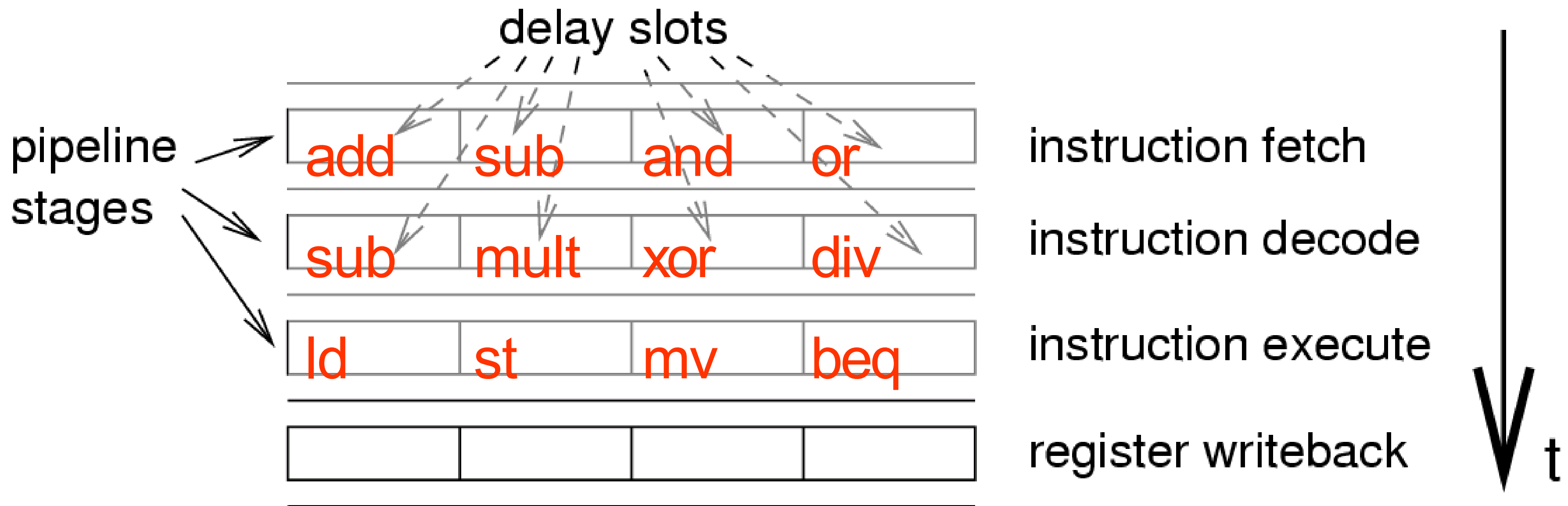
register writeback



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.

© p.marwedel, informatik 12, 2008

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:
```

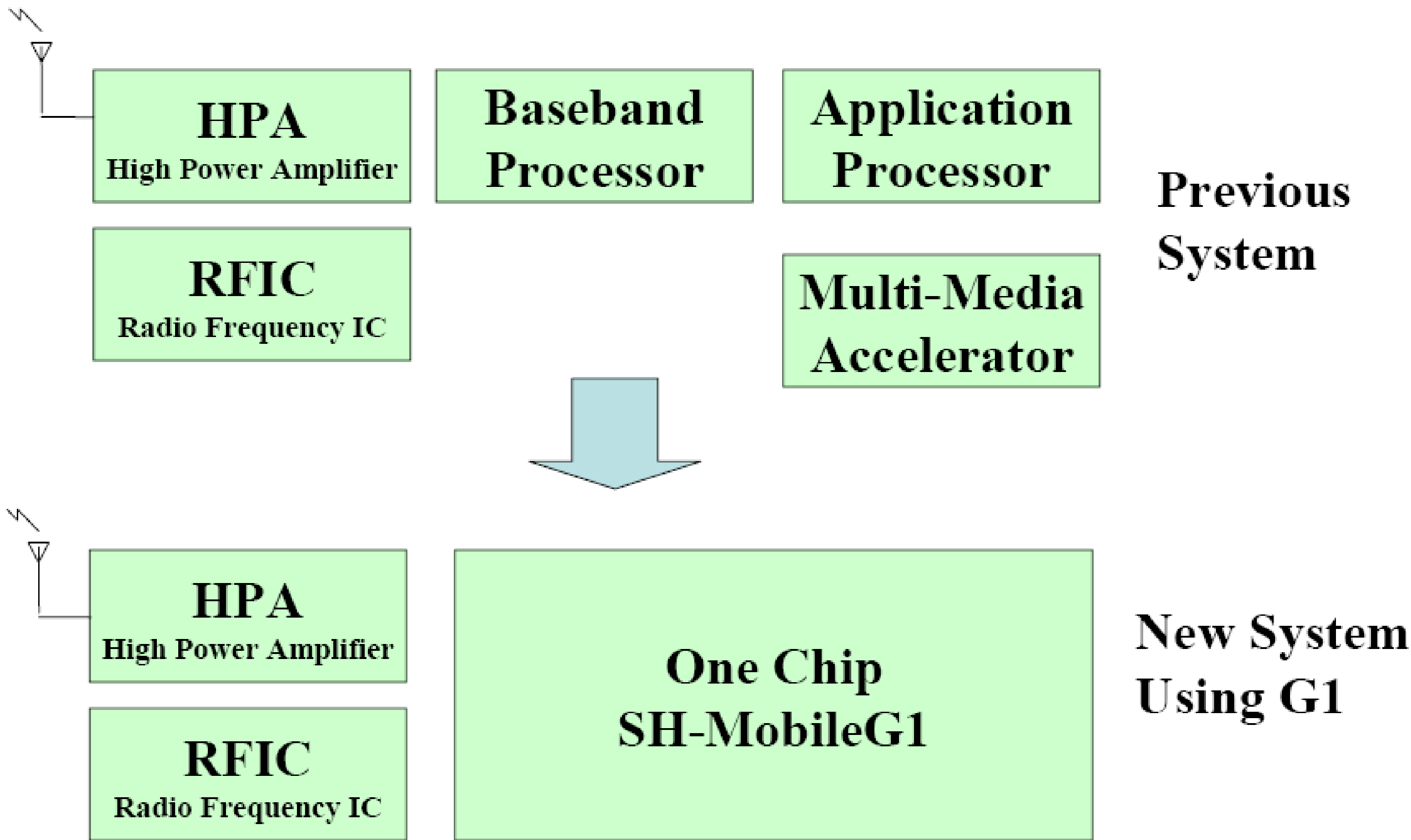
max. 12 cycles

Predicated execution

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

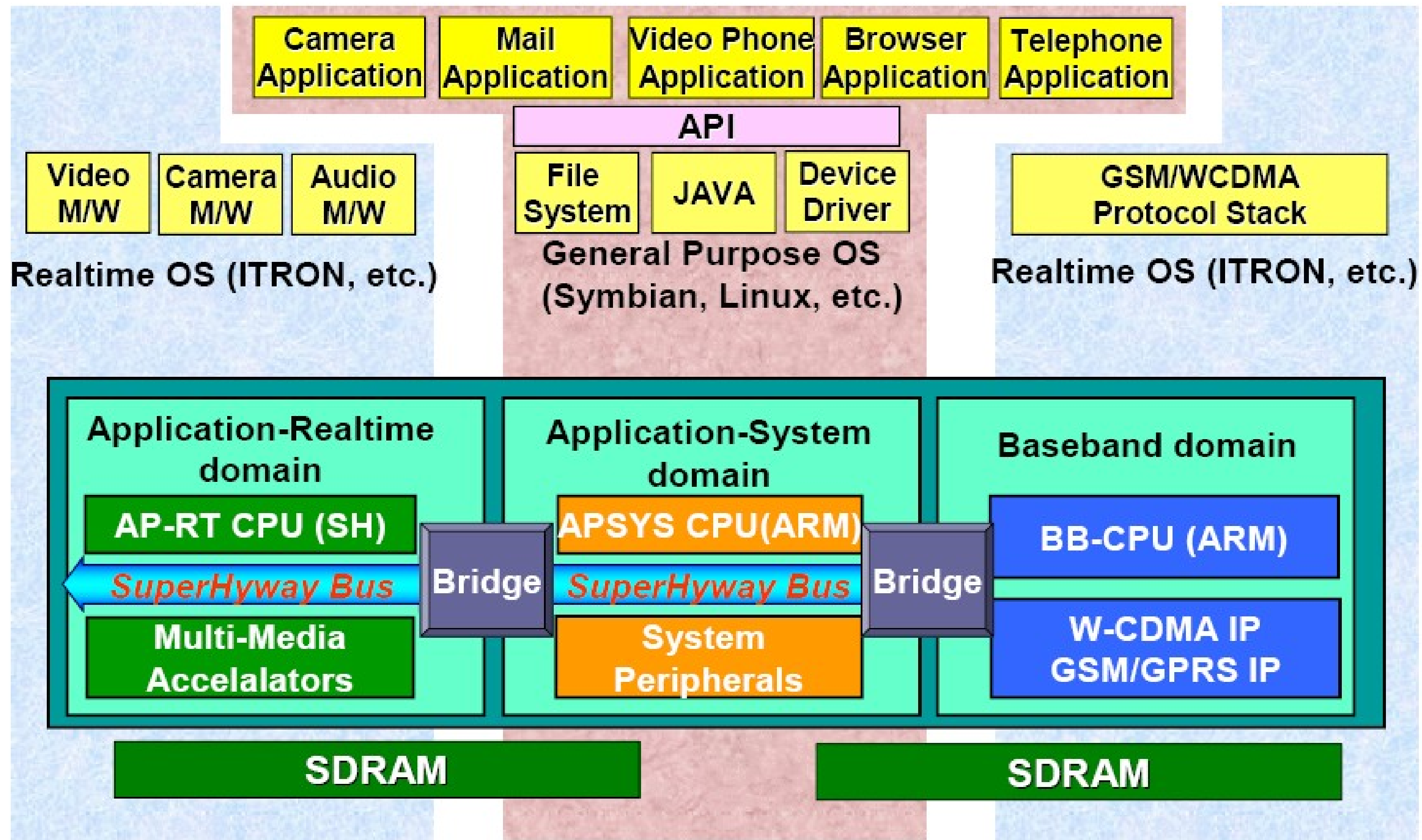
1 cycle

Trend: multiprocessor systems-on-a-chip (MPSoCs)



http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

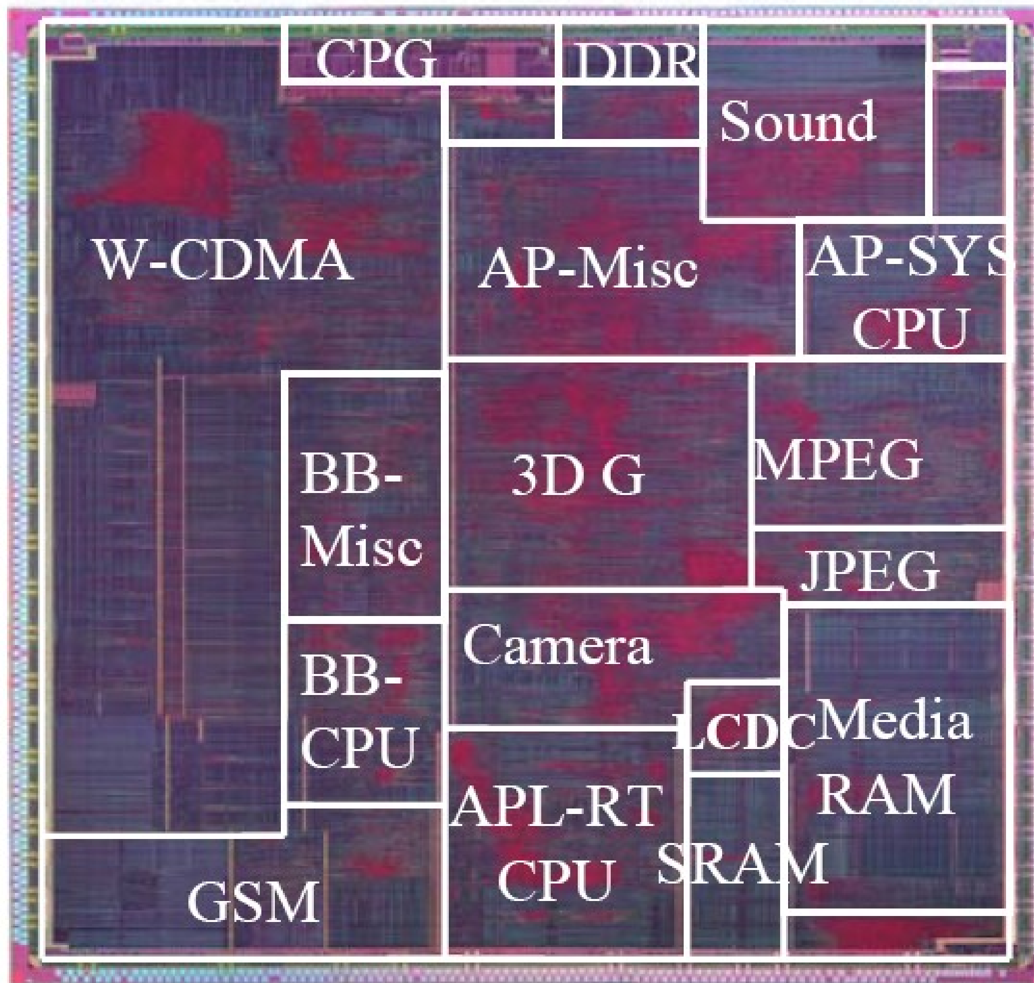
Multiprocessor systems-on-a-chip (MPSoCs) (2)



http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

Multiprocessor systems-on-a-chip (MPSoCs) (3)

SH-MobileG1: Chip Overview



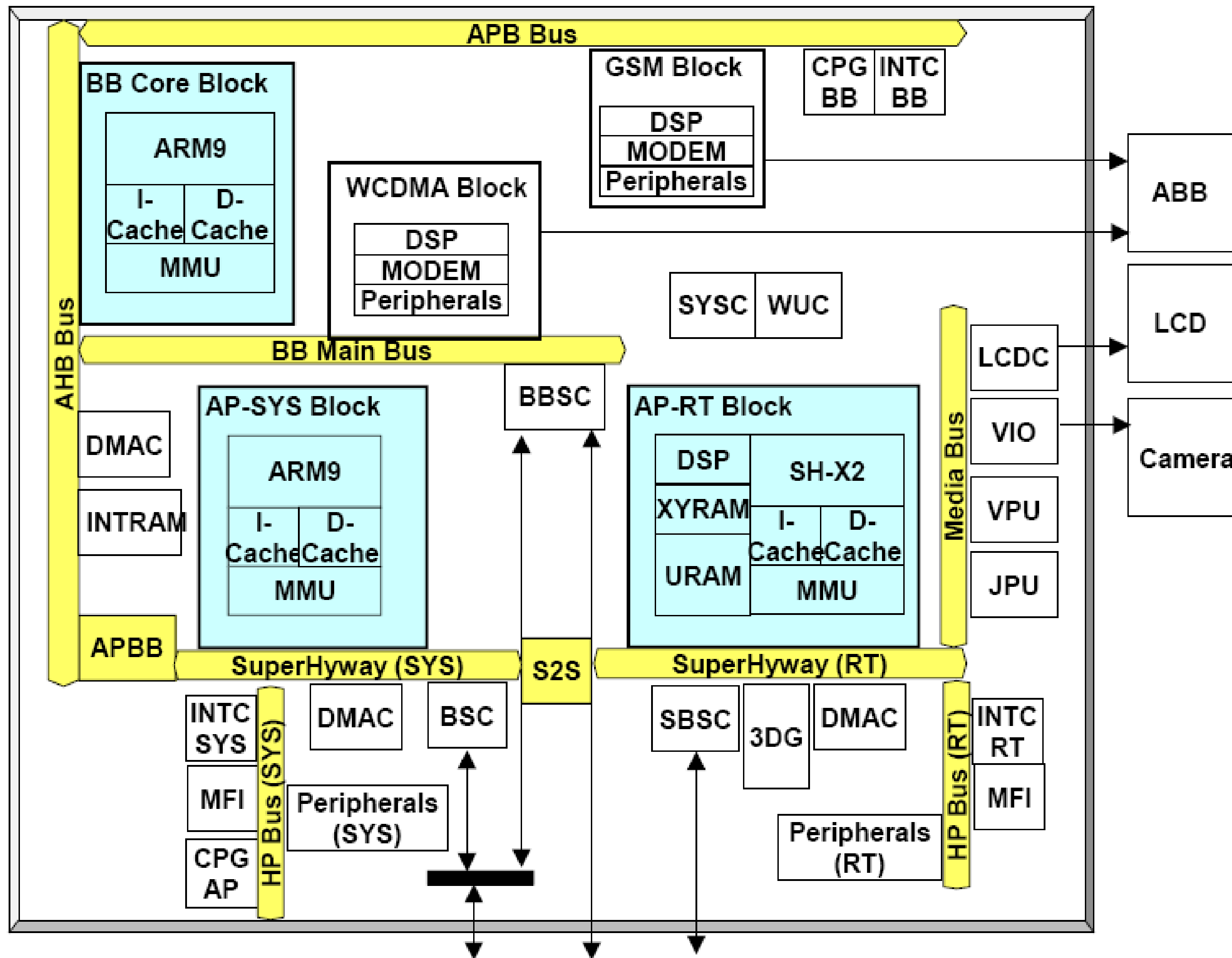
Die size	11.15mm x 11.15mm
Process	90nm LP 8M(7Cu+1Al) CMOS dual-Vth
Supply voltage	1.2V(internal), 1.8/2.5/3.3V(I/O)
# of TRs, gate, memory	181M TRs, 13.5M Gate 20.2 Mbit mem

http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

©2007. Renesas Technology Corp., All rights reserved.

Multiprocessor systems-on-a-chip (MPSoCs) (4)

G1 Module Diagram



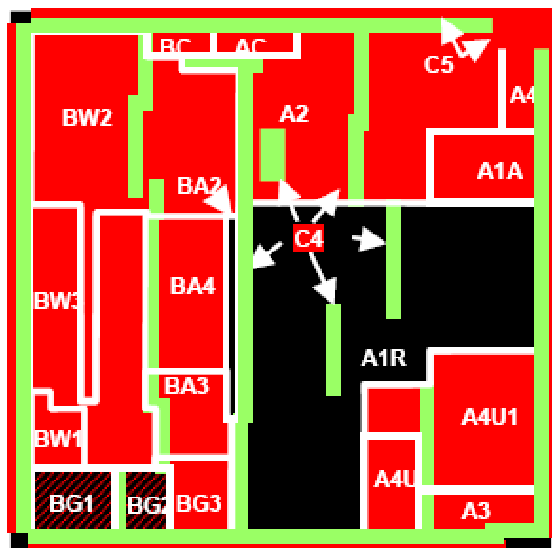
http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

 ©2007. Renesas Technology Corp., All rights reserved. Everywhere you imagine. RENESAS

Multiprocessor systems-on-a-chip (MPSoCs) (5)

Leakage Current in Usage Scenes

(2)Telephony (W-CDMA)



Baseband part	Control	ON
	W-CDMA	ON
	GSM	ON / OFF
Application part	System-domain	ON
	Realtime-domain	OFF
Measured Leakage Current (@ Room Temp, 1.2V)		407 μA

http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

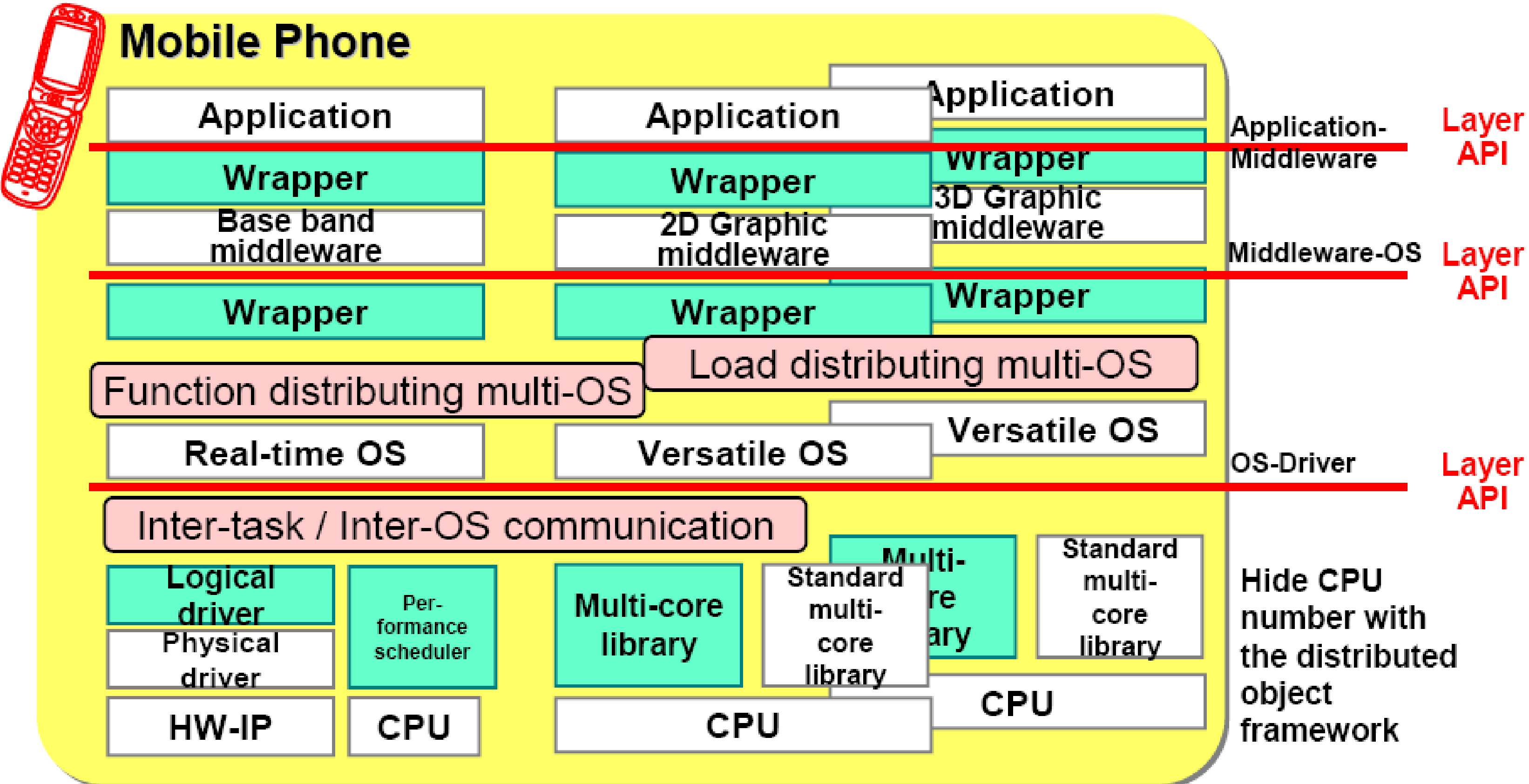
©2007. Renesas Technology Corp., All rights reserved.

Everywhere you imagine. RENESAS

Multiprocessor systems-on-a-chip (MPSoCs) (6)

EXREAL Platform™ Software Interconnect

- Promote reuse of software assets through Wrapper and standardized layer API
- Control operating frequency and power on/off through the performance scheduler



Hide CPU number with the distributed object framework

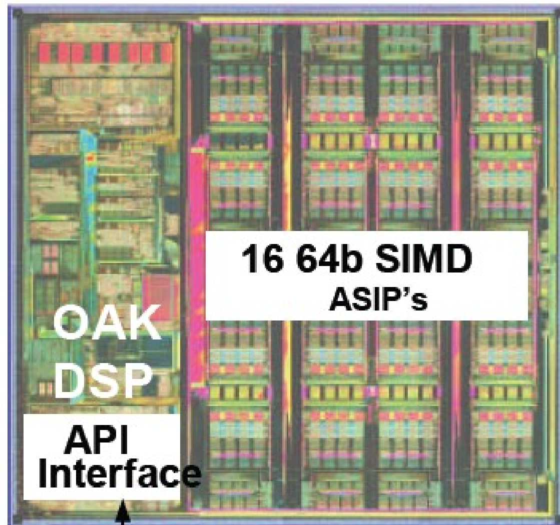
http://www.mpsoc-forum.org/2007/slides/Hattori.pdf

©2007. Renesas Technology Corp., All rights reserved.

Everywhere you imagine. RENESAS

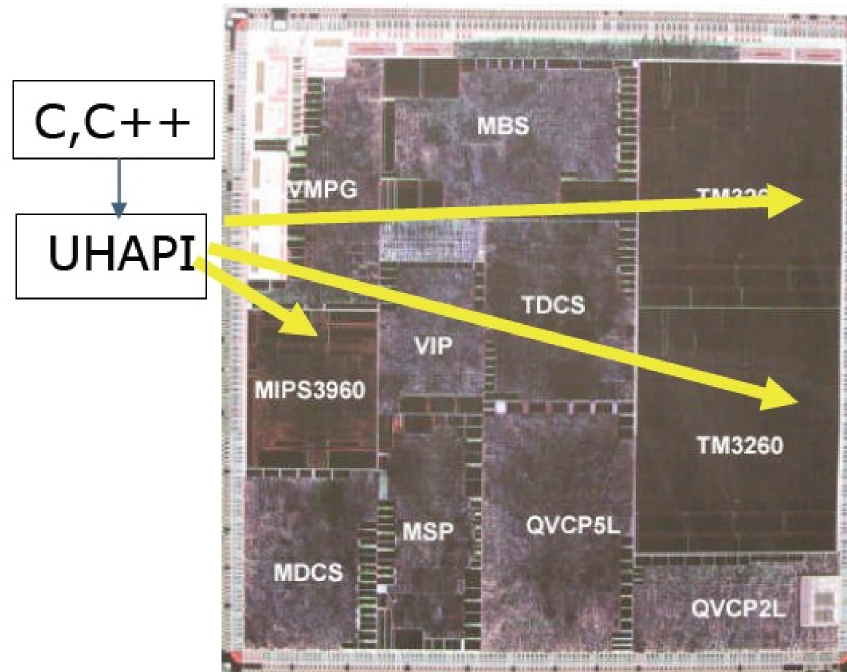
Multiprocessor systems-on-a-chip (MPSoCs) (7)

VIP for car mirrors
Infineon



200MHz , 0.76 Watt
100Gops @ 8b
25Gops @ 32b

Nexperia Digital Video Platform
NXP



1 MIPS, 2 Trimedia
60 coproc, 250 RAM's
266MHz, 1.5 watt 100 Gops

~50% inherent power efficiency of silicon

© Hugo De Man, IMEC, 2007

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