طراحی سیستم‌های تعبیه‌شده

Embedded System Design

فصل سوم ـ قسمت دوم

سخت‌افزار سیستم تعبیه‌شده

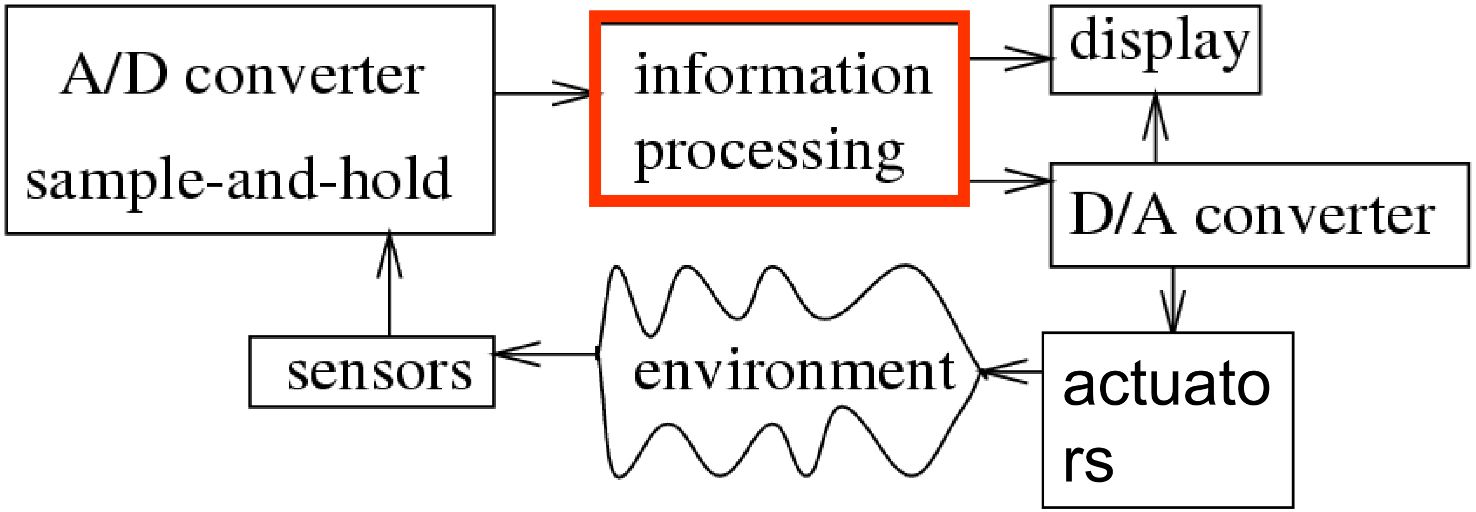
Embedded System Hardware

کاظم فولادی
دانشکده‌ی مهندسی برق و کامپیوتر
دانشگاه تهران

kazim@fouladi.ir

Embedded System Hardware

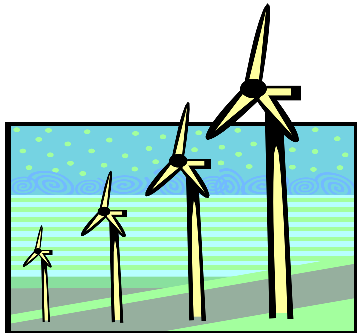
سخت‌افزار سیستم تعبیه‌شده معمولاً در یک حلقه (loop) استفاده می‌شود
:(*“hardware in a loop”*)



واحدهای پردازش
Processing units

نیاز به کارامدی (توان + انرژی):

چرا نگران انرژی و توان هستیم؟



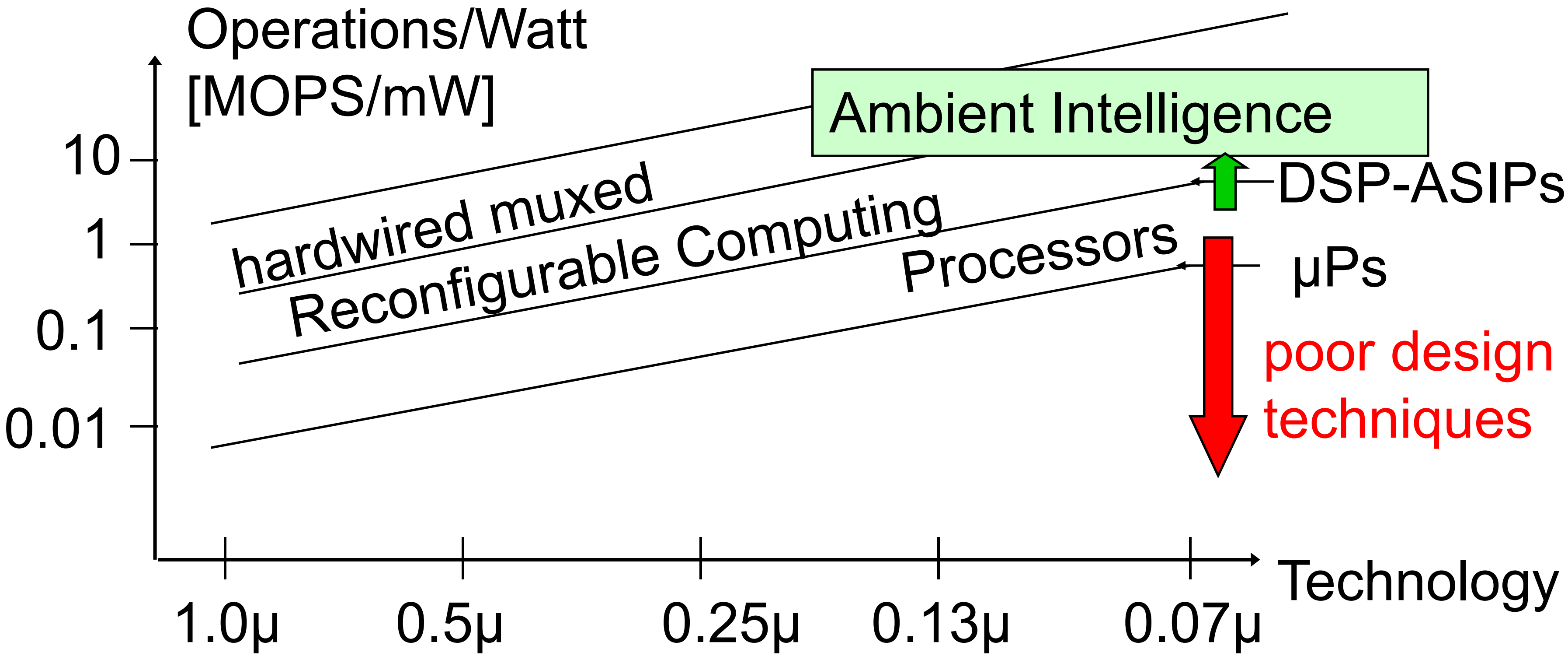
«توان به عنوان مهم‌ترین محدودیت در سیستم‌های تعبیه‌شده در نظر گرفته می‌شود»

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

Current UMTS phones can hardly be operated for more than an hour, if data is being transmitted.

[from a report of the Financial Times, Germany, on an analysis by Credit Suisse First Boston; http://www.ftd.de/tm/tk/9580232.html?nv=se]

تداخل انرژی/انعطاف‌پذیری ـ کارامدی توان داخلی ـ
The energy/flexibility conflict - Intrinsic Power Efficiency -

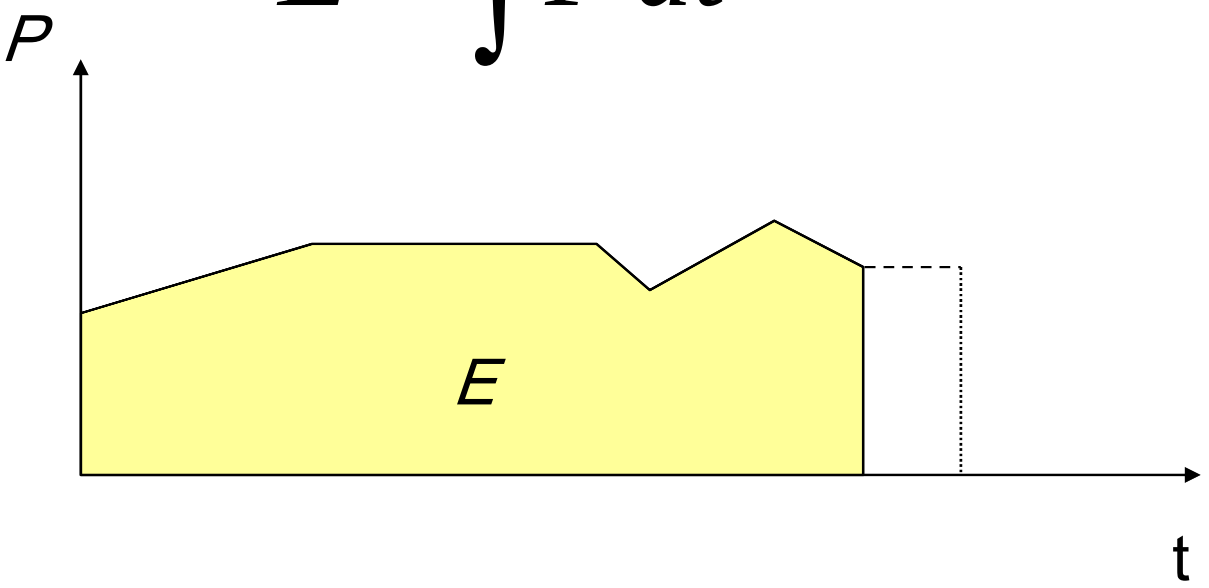


لزوم بهینه‌سازی سخت‌افزار/نرم‌افزار؛ در غیر این صورت هزینه‌ی انعطاف‌پذیری نرم‌افزار قابل پرداخت نیست!

[H. de Man, Keynote, DATE‘02; T. Claasen, ISSCC99]

توان و انرژی به همدیگر مرتبط هستند
Power and energy are related to each other

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



در بسیاری از موارد اجرای سریع‌تر به معنی انرژی کمتر است، اما مخالف آن نیز ممکن است درست باشد: اگر لازم باشد توان برای امکان اجرای سریع‌تر افزایش یابد.

مصرف توان کم در مقابل مصرف انرژی کم
Low Power vs. Low Energy Consumption

- می‌نیمم کردن **توان مصرفی** مهم است برای
  - طراحی منبع توان
  - طراحی رگولاتور ولتاژ
  - تعیین ابعاد اتصالات میانی
  - خنک‌سازی کوتاه‌مدت
- می‌نیمم کردن **انرژی مصرفی** مهم است، به دلیل
  - محدود بودن وجود انرژی (سیستم‌های متحرک)
    - ظرفیت‌های محدود باتری (بهبود فناوری آن با سرعت کم)
    - هزینه‌ی بسیار بالای انرژی
  - خنک‌سازی
    - هزینه‌ی بالا
    - فضای محدود
  - قابلیت اتکا
  - طول عمر زیاد، دمای پایین

مدارهای با کاربرد خاص

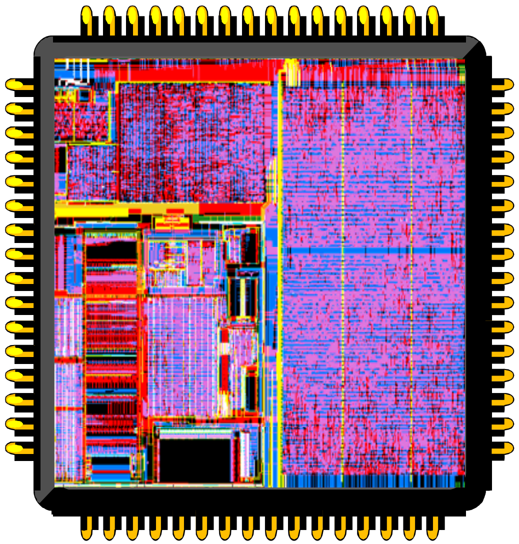
Application Specific Circuits (ASICs) or Full Custom Circuits

لزوم مدارهای طراحی شده به طور سفارشی

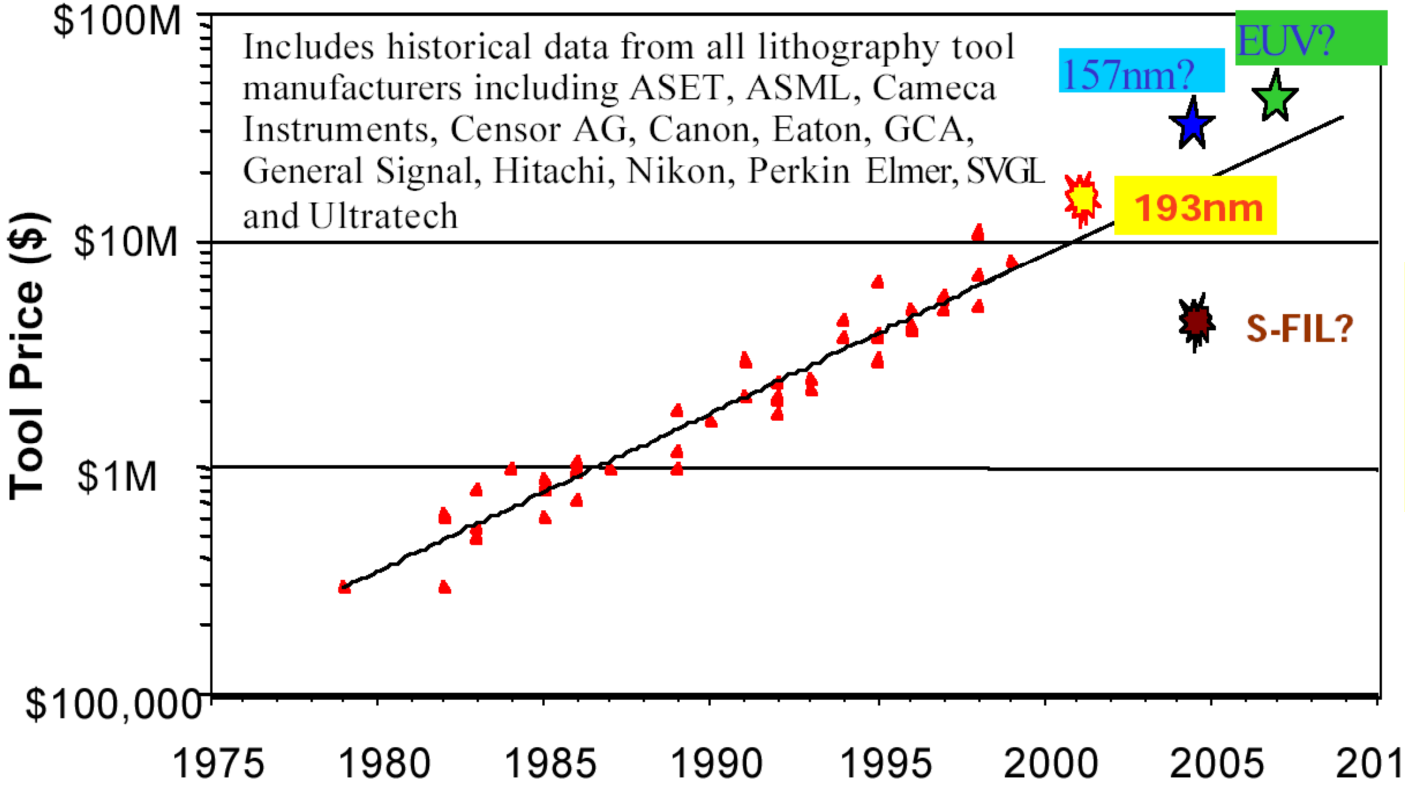
- نیاز به حداکثر سرعت
- با هدف کارامدی انرژی
- فروش تعداد زیادی از آن تراشه

مشکلات این رهیافت:

- زمان طولانی برای طراحی
- عدم وجود انعطاف‌پذیری (تغییر استانداردها) و
- هزینه‌ی بالا (برای هر ماسک در حدود چند میلیون دلار)



هزینه‌ی ماسک برای سخت‌افزار اختصاصی بسیار گران می‌شود
Mask cost for specialized HW becomes very expensive

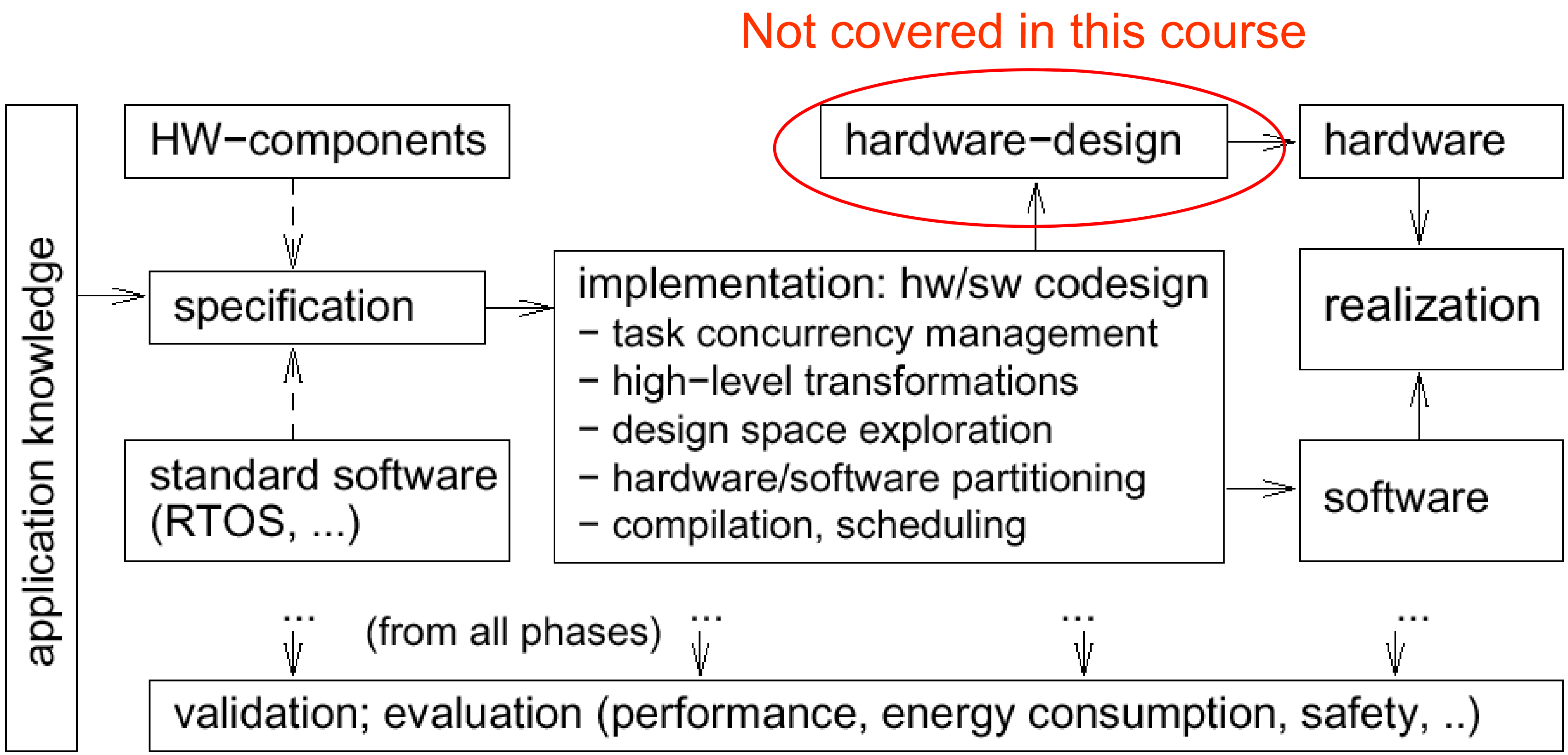


تمایل:
به سوی
پیاده‌سازی در
نرم‌افزار

ASIC synthesis not covered in this course.

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

Structure of this course



پردازنده‌ها
Processors

At the chip level, embedded chips include micro-controllers and microprocessors. ***Micro-controllers*** *are the true* ***workhorses*** *of the embedded family. They are the original 'embedded chips' and include those first employed as controllers in elevators and thermostats* [Ryan, 1995].

Microcontrollers
- MHS 80C51 as an example -

- 8-bit CPU optimized for control applications
- Extensive Boolean processing capabilities
- 64 k Program Memory address space
- 64 k Data Memory address space
- 4 k bytes of on chip Program Memory
- 128 bytes of on chip data RAM
- 32 bi-directional and individually addressable I/O lines
- Two 16-bit timers/counters
- Full duplex UART
- 6 sources/5-vector interrupt structure with 2 priority levels
- On chip clock oscillators
- Very popular CPU with many different variations

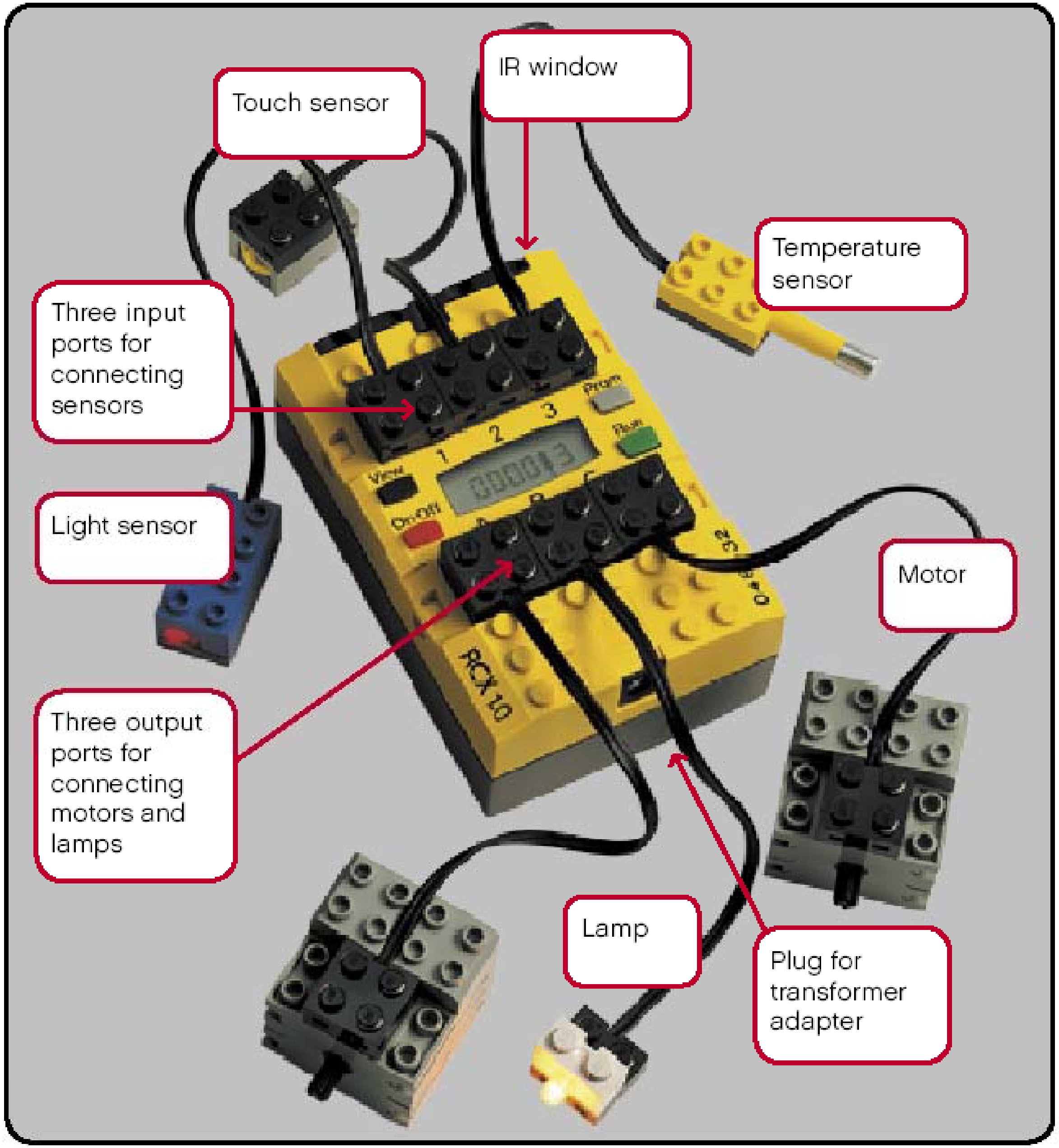
Features for Embedded Systems

More in-depth:

Microcontrollers and the Lego® Mindstorm Lab

www.watch.impress.co.jp/.../20000821/minds.htm

RCX, the Lego® control unit



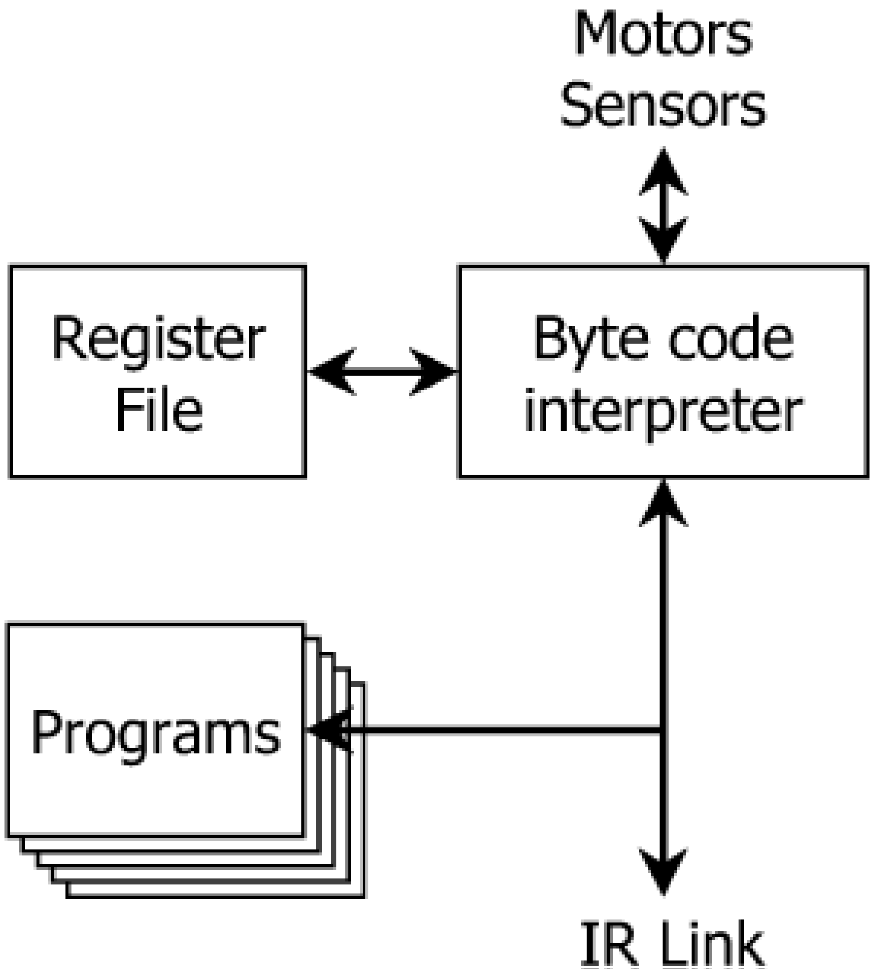
Salient features of the RCX

- The RCX has a piezoelectric speaker, which produces 6 distinct tones and can even 'carry a tune'.
- Three input ports: Three gray 4-stud bricks above LCD labeled 1, 2, 3.
- Three output ports: Three black 4-stud bricks below LCD labeled, A, B, C.
- Four control buttons (View, On-Off, Prgm, Run).
- LCD display.
- Using infrared communication, the RCX can:
  - communicate with a computer
  - communicate with other RCX bricks: messages can be passed from RCX to RCX
  - be controlled via the Remote Control

RCX implements virtual byte code machine

Opcode	Modes			Name	Encoding
10/18	P			Alive	void
12/1a	P			Get value	byte *source* byte *argument*
13/1b	P		C	Set motor power	byte *motors* byte *source* byte *argument*
14/1c	P		C	Set variable	byte *index* byte *source* short *argument*
15/1d	P			Get versions	byte *key*[5]
16/1e		R		Set motor direction	void
17/xx			C	Call subroutine	byte *subroutine*
20/28	P			Get memory map	void
21/29	P		C	Set motor on/off	byte *code*

The RCX Virtual Machine



- Interpreter executes byte code from two sources
- Up to five programs, each consisting of:
  - up to 10 tasks
  - up to 8 subroutines
- Memory map stores locations of tasks and subroutines

http://graphics.stanford.edu/~kekoa/rcx/talk/talk.018.html

EE380 Lecture

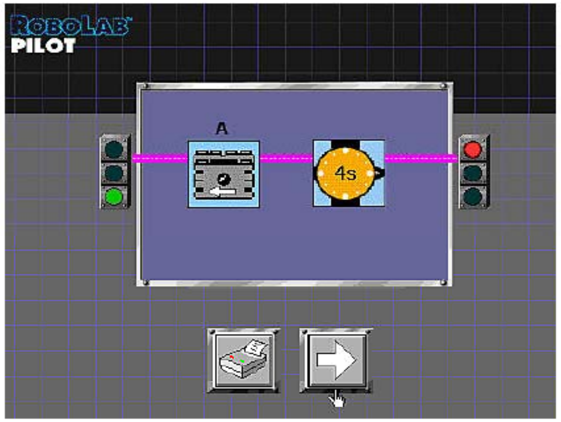
1. Graphical user interface RoboLab

Pilot Level 1

Turning a motor on or off

1. Connect a motor to port A on your RCX and turn the RCX on by pressing the red On-Off button. If you connect a wheel to the motor you will be able to see which direction the motor is programmed to run.

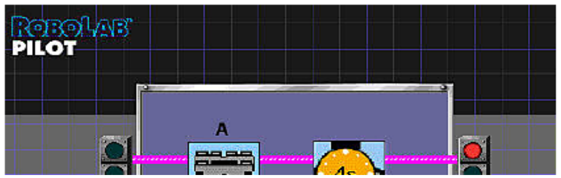
2. Start ROBOLAB, select Programmer and double-click on Pilot 1. A default program will appear on your screen. The motor icon offers you a left (clockwise) or right (counter clockwise) option.



3. Place your RCX in front of the IR Tower. Make sure the RCX is turned on. NOTE that the RCX automatically turns off after 15 minutes.

4. Select the white arrow button, which is the download button. A new box appears on your screen indicating that download is proceeding.

5. Press the green Run button on your RCX.
a. Is the motor running? If not—have you connected the wire to port A?



2. Textual user interface NQC (Not quite C)

C-like programs translated into CRX-bytecode

Composed of:

1. Global variables
2. Task blocks
3. Inline functions
4. subroutines

Tasks

```
task name( )
{
    // the task 's code is placed here
}
```

name: any legal identifier.

1 task - named "main" - started when program is run.

Maximum number of tasks on RCX: 10

The body of a task consists of a list of statements.

Tasks started and stopped: `start` and `stop` statements

`StopAllTasks` stops all currently running tasks.

(Inline) Functions

void *name*(*argument_list*)
{ // body of the function }
Functions cannot return a value; void is related to C
Argument list: empty, or ≥ 1 argument definitions.
Arguments: *type* followed by its *name*.
All values are 16 bit signed integers.
4 different argument classes:

Type	Meaning	Restriction
int	Pass by value	None
int&	Pass by reference	Only variables may be used
const int	Pass by value	Only constant may be used
const int&	Pass by reference	Function cannot modify argument

Subroutines

Subroutines allow a single copy of some code to be shared between several different callers (space efficient).

Restrictions:

- First of all, subroutines cannot use any arguments.
- A subroutine cannot call another subroutine.
- Maximum number of subroutines: 8 for the RCX
- If calling from multiple tasks: no local variables or perform calculations that require temporary variables (this restriction is lifted for the Scout and RCX2).

Variables

All variables of the same type: 16 bit signed integers.

Declarations:

`int` variable[=initial value] [, variable [=initial value]] ;

Examples:

```
int x ;          // declare x
int y, z ;       // declare y and z
int a =1, b ; // declare a and b, initialize a to 1
```

- Global variables: declared at the program scope; Used within tasks, functions, subroutines. Max: 32
- Local variables: within tasks, functions, and sometimes within subroutines. Max: 0 @ RCX, 16 @RCX2
- Local variables may be declared in a compound statement, following a {

Arrays

Arrays exist only for RCX2

Assignments

Syntax:

Variable operator expression

Operators:

=	Set variable to expression
+=	Add expression to variable
-=	Subtract expression from variable
*=	Multiple variable by expression
/=	Divide variable by expression
&=	Bitwise AND expression into variable
|=	Bitwise OR expression into variable
||=	Set variable to absolute value of expression
+-=	Set variable to sign (-1,+1,0) of expression

Control structures

- **If-statements**
  if (*condition*) *consequence*
  if (*condition*) *consequence* **else** *alternative*
- **While-statements**
  while (*condition*) *body*
- **Repeat-statements**
  repeat (*expression*) *body*
- **Switch-statement**
  switch (*expression*) *body*
- **Until-macro**
  # **define** until (c) **while** (! (c))

Built-in API

SetPower(outputs, power) **Function**
Sets the power level of the specified outputs.
Power should result in a value between 0 and 7.
OUT_LOW, OUT_HALF, OUT_FULL may also be used.
Examples:
SetPower(OUT_A, OUT _FULL) ; // A full power
SetPower(OUT_B, x);

OnFwd(outputs) **Function**
Set outputs to forward direction and turn them on.
Outputs is one or more of OUT_A, OUT_B, and OUT_C added together.
Example: OnFwd (OUT_A);

Sensor Types

Sensor Type	Meaning
SENSOR_TYPE_NONE	generic passive sensor
SENSOR_TYPE_TOUCH	a touch sensor
SENSOR_TYPE_TEMPERATURE	a temperature sensor
SENSOR_TYPE_LIGHT	a light sensor
SENSOR_TYPE_ROTATION	a rotation sensor

Sensor Modes

Sensor Mode	Meaning
SENSOR_MODE_RAW	raw value from 0 to 1023
SENSOR_MODE_BOOL	boolean value (0 or 1)
SENSOR_MODE_EDGE	counts number of boolean transitions
SENSOR_MODE_PULSE	counts number of boolean periods
SENSOR_MODE_PERCENT	value from 0 to 100
SENSOR_MODE_FAHRENHEIT	degrees F - RCX only
SENSOR_MODE_CELSIUS	degrees C - RCX only
SENSOR_MODE_ROTATION	rotation (16 ticks per revolution) - RCX only

Sensor Type/Mode Combinations

Sensor Configuration	**Type**	**Mode**
SENSOR_TOUCH	SENSOR_TYPE_TOUCH	SENSOR_MODE_BOOL
SENSOR_LIGHT	SENSOR_TYPE_LIGHT	SENSOR_MODE_PERCENT
SENSOR_ROTATION	SENSOR_TYPE_ROTATION	SENSOR_MODE_ROTATION
SENSOR_CELSIUS	SENSOR_TYPE_TEMPERATURE	SENSOR_MODE_CELSIUS
SENSOR_FAHRENHEIT	SENSOR_TYPE_TEMPERATURE	SENSOR_MODE_FAHRENHEIT
SENSOR_PULSE	SENSOR_TYPE_TOUCH	SENSOR_MODE_PULSE
SENSOR_EDGE	SENSOR_TYPE_TOUCH	SENSEO_MODE_EDGE

Setting Sensor Type and Mode

SetSensor(sensor, configuration)
Set the type and mode to the specified configuration (constant containing both type and mode info).
Example:
SetSensor (SENSOR_1, SENSOR_TOUCH) ;

SetSensorType(sensor, type)
Set type (one of the predefined sensor type constants).
Example:
SetSensorType(SENSOR_1, SENSOR_TYPE _TOUCH);

SetSensorMode(sensor, mode)
Set mode (one of the predefined sensor mode constants)
Optional slope parameter for Boolean conversion.

Reading out sensors, Wait

SensorValue(n)

Returns the processed sensor reading for sensor n, where n is 0, 1, or 2. This is the same value that is returned by the sensor names (e.g. SENSOR_1).

Example:

```
x = SensorValue(0);  // readsensor_1
```

Wait(time)

Make a task sleep for specified amount of time (in 1/100 s). Argument may be an expression or a constant.

```
Wait(100);                // wait 1 second
Wait(Random (100)); // wait random time up to 1 second
```

For more information refer to the NQC programmers manual

Example

```
// speed.nqc -- sets motor power, goes forward, waits,
// goes backwards
task main()
{
  SetPower(OUT_A+OUT_C,2);
  OnFwd(OUT_A+OUT_C);
  Wait(400);
  OnRev(OUT_A+OUT_C);
  Wait(400);
  Off(OUT_A+OUT_C);
}
```

Spiral

```
// spiral.nqc -- Uses repeat & variables to make robot
// move in a spiral
#define TURN_TIME   100
int move_time;                // define a variable
task main()
{  move_time = 20;            // set the initial value
  repeat(50)
  { OnFwd(OUT_A+OUT_C);
    Wait(move_time);       // use the variable for sleeping
    OnRev(OUT_C);
    Wait(TURN_TIME);
    move_time += 5;        // increase the variable
  }  Off(OUT_A+OUT_C); }
```

Use of touch sensors

```
// Use of touch sensors
task main()
{ SetSensor(SENSOR_1,SENSOR_TOUCH);
  OnFwd(OUT_A+OUT_C);
  while (true)
  { if (SENSOR_1 == 1)
    { OnRev(OUT_A+OUT_C); Wait(30);
      OnFwd(OUT_A); Wait(30);
      OnFwd(OUT_A+OUT_C);
    }
  }
}
```

Use of light sensor

```
// Use of a light sensor to make robot go forward until
// it "sees" black, then turn until it's over white
#define THRESHOLD 37
task main()
{ SetSensor(SENSOR_2,SENSOR_LIGHT);
  OnFwd(OUT_A+OUT_C);
  while (true)
  { if (SENSOR_2 < THRESHOLD)
    { OnRev(OUT_C);
      Wait(10);
      until (SENSOR_2 >= THRESHOLD);
      OnFwd(OUT_A+OUT_C);
    } } }
```

Tasking

```
task main()
{ SetSensor(SENSOR_1,SENSOR_TOUCH);
  start check_sensors;
  start move_square; }
task move_square()
{ while (true)
  { OnFwd(OUT_A+OUT_C); Wait(100);
    OnRev(OUT_C); Wait(85);  } }
task check_sensors()
{ while (true)
  { if (SENSOR_1 == 1)
    { stop move_square;
      OnRev(OUT_A+OUT_C); Wait(50);
      OnFwd(OUT_A); Wait(85);
      start move_square; } } }
```

Subroutines

```
sub turn_around()
{ OnRev(OUT_C); Wait(400);
  OnFwd(OUT_A+OUT_C);
}
task main()
{ OnFwd(OUT_A+OUT_C);
  Wait(100);
  turn_around();
  Wait(200);
  turn_around();
  Wait(100);
  turn_around();
  Off(OUT_A+OUT_C);}
```

Inline function, call by reference

```
void turn_around(int turntime)
{ OnRev(OUT_C); Wait(turntime);
  OnFwd(OUT_A+OUT_C); }
task main()
{ OnFwd(OUT_A+OUT_C);
  Wait(100);
  turn_around(200);
  Wait(200);
  turn_around(50);
  Wait(100);
  turn_around(300);
  Off(OUT_A+OUT_C); }
```

```
task main()
{  int count=0;
   while (count<=5)
   {  PlaySound(SOUND_CLICK);
      Wait(count*20);
      increment(count);
   }
}
void increment(int& n)
{ n++; }
```

Playing preprogrammed sounds & tones

```
task main()
{ PlaySound(0); Wait(100);
  PlaySound(1); Wait(100);
  PlaySound(2); Wait(100);
  PlaySound(3); Wait(100);
  PlaySound(4); Wait(100);
  PlaySound(5); Wait(100);
}
```

```
task music()
{ while (true)
  { PlayTone(262,40);  Wait(50);
    PlayTone(294,40);  Wait(50);
    PlayTone(330,40);  Wait(50);
    PlayTone(294,40);  Wait(50);
  } }
task main()
{ start music;
  while(true)
  { OnFwd(OUT_A+OUT_C); Wait(300);
    OnRev(OUT_A+OUT_C); Wait(300);
  } }
```

Macros

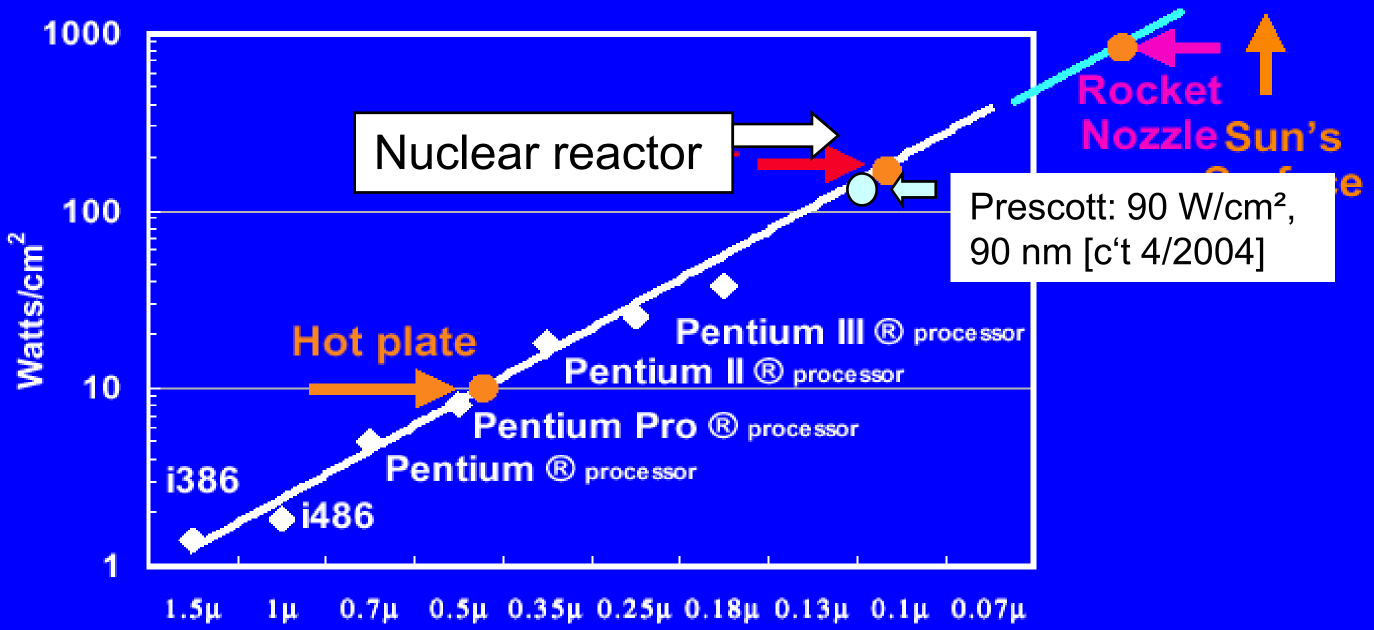
end

```
#define turn_right(s,t)
SetPower(OUT_A+OUT_C,s);OnFwd(OUT_A);OnRev(OUT_C);Wait(t);
#define turn_left(s,t)
SetPower(OUT_A+OUT_C,s);OnRev(OUT_A);OnFwd(OUT_C);Wait(t);
#define forwards(s,t)
SetPower(OUT_A+OUT_C,s);OnFwd(OUT_A+OUT_C);Wait(t);
#define backwards(s,t)
SetPower(OUT_A+OUT_C,s);OnRev(OUT_A+OUT_C);Wait(t);
task main()
{ forwards(1,200);   turn_left(7,85);
  forwards(4,100);   backwards(1,200);
  forwards(7,100);   turn_right(4,85);
  forwards(1,200);   Off(OUT_A+OUT_C);}
```

نیازمندی‌های کلیدی برای پردازنده‌ها

۱. کارامدی انرژی/توان

Power density continues to get worse



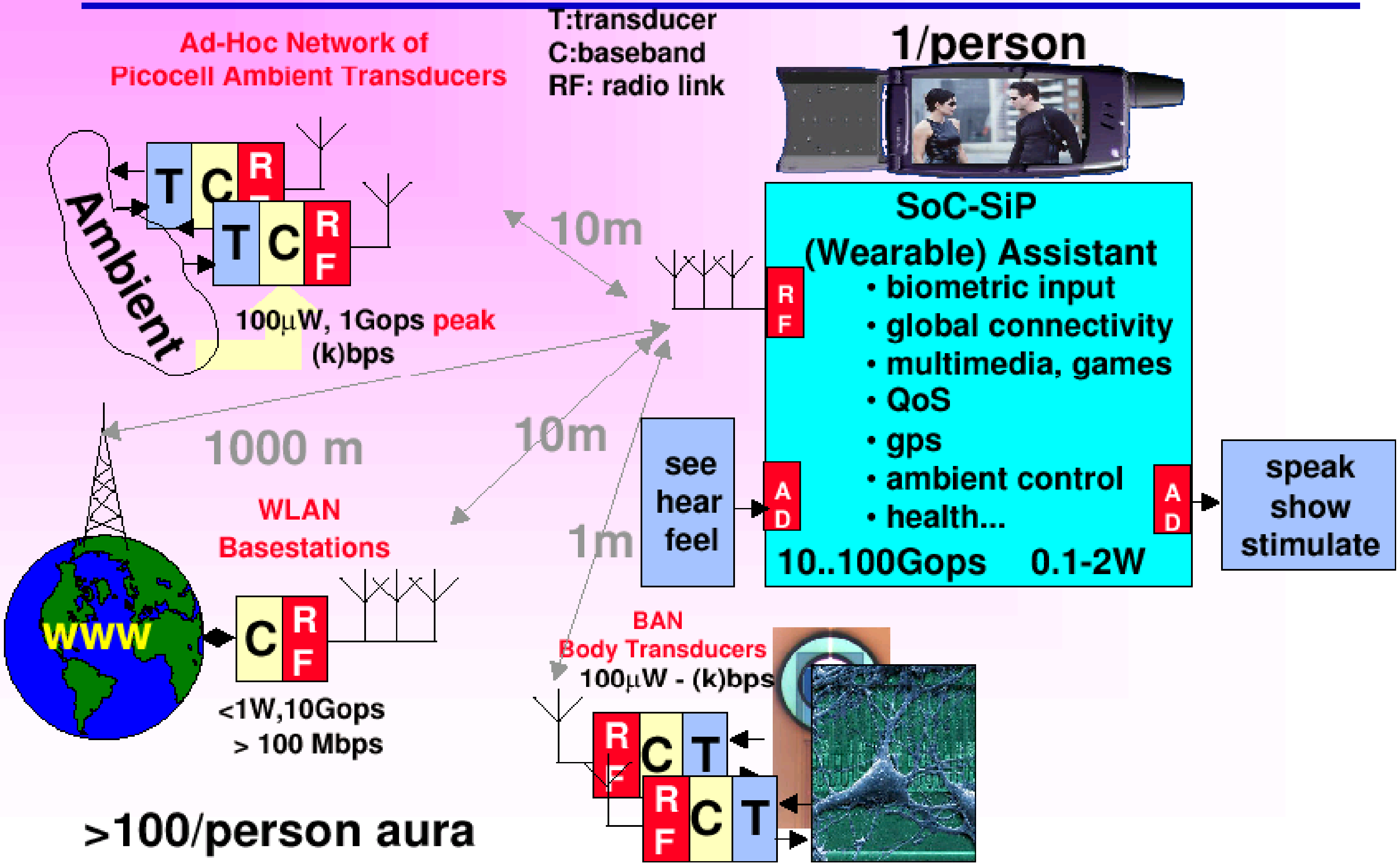
Surpassed hot-plate power density in 0.5µ

Not too long to reach nuclear reactor

Micro32
Fred Pollack

intel

Ambient Intelligence Global System



© H.De Man/IMEC DATE02 7

Nano-systems with Giga-complexity

Need global system optimisation
GHz RF and mixed signal everywhere

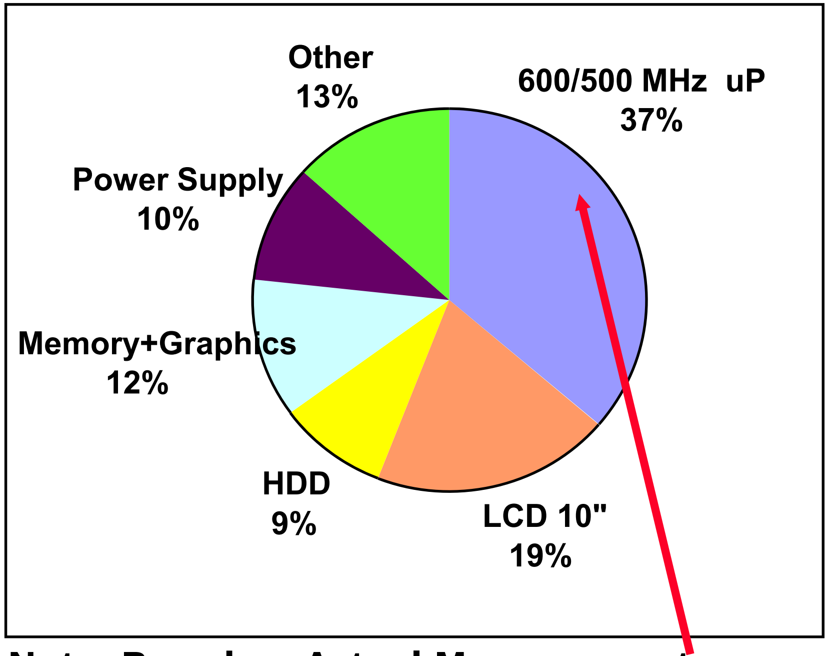
Transducer nodes	Assistant nodes/basestations
☛ Ultra low energy (100Mops/mW) ☛ Ultra low cost (1€) ☛ Low flexibility ☛ 1..10 Mtr (small size) ☛ DSP&RF dominated ☛ Chip-package co-design ☛ Ultra fast hw design	☛ Low energy (10-50Mops/mW) ☛ Low cost (100 €) ☛ High Flexibility ☛ 10..100 Gops, >100 Mtr ☛ Data-Intensive, dynamic tasks ☛ Task and data concurrency ☛ Incremental sw design
"ASIC in a week"	"PLATFORM"

@100..1000 times Power efficiency of today's μP...

 11

Need to consider CPU & System Power

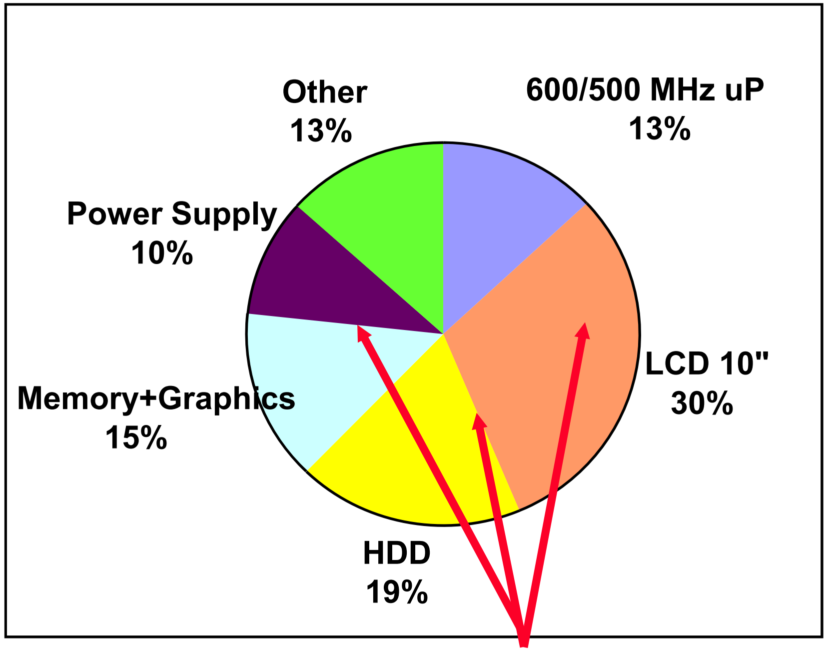
Mobile PC
Thermal Design (TDP) System Power



Note: Based on Actual Measurements

CPU Dominates Thermal Design Power

Mobile PC
Average System Power

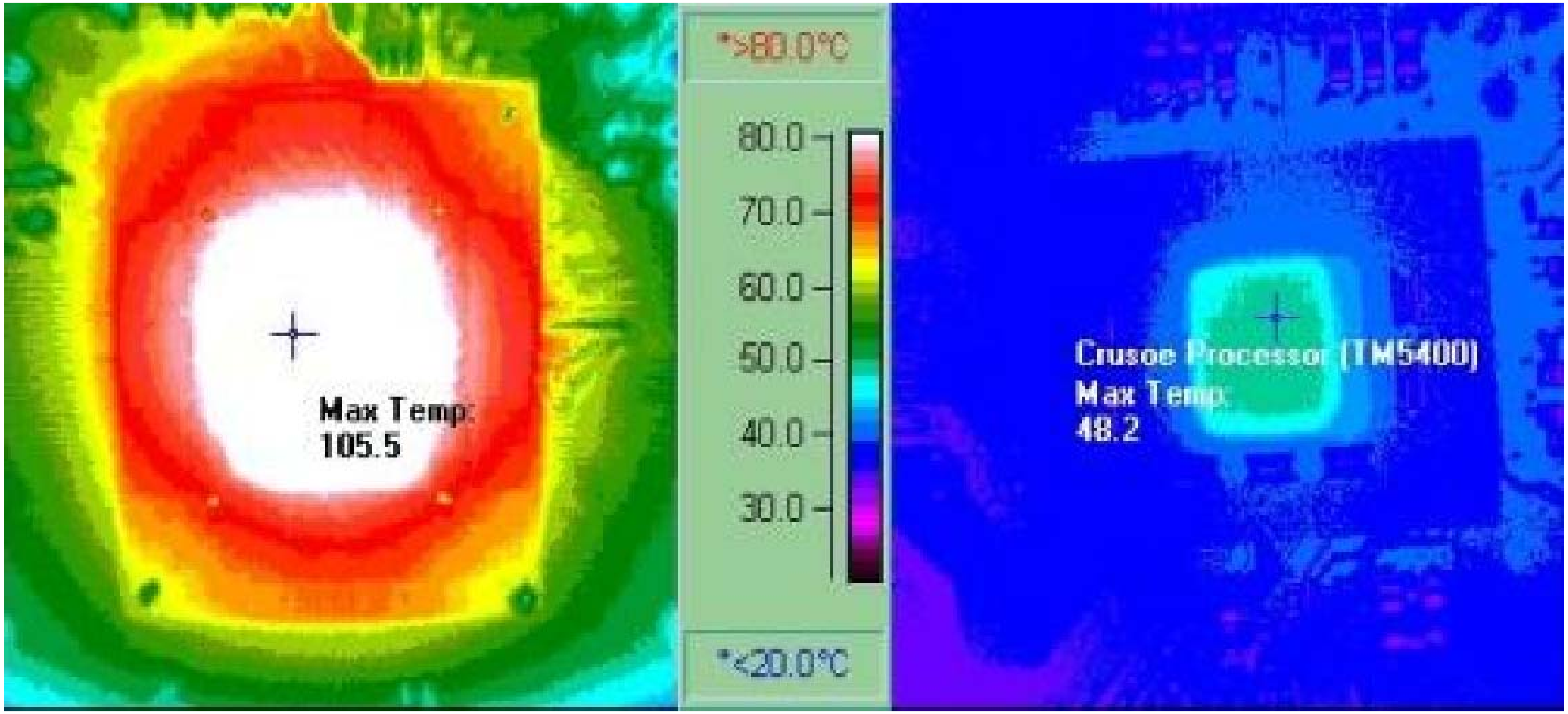


Multiple Platform Components Comprise Average Power

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

ایده‌های جدید واقعاً می‌تواند مصرف انرژی را کاهش دهد
New ideas can actually reduce energy consumption

Pentium | Crusoe



Running the same multimedia application.

As published by Transmeta [www.transmeta.com]

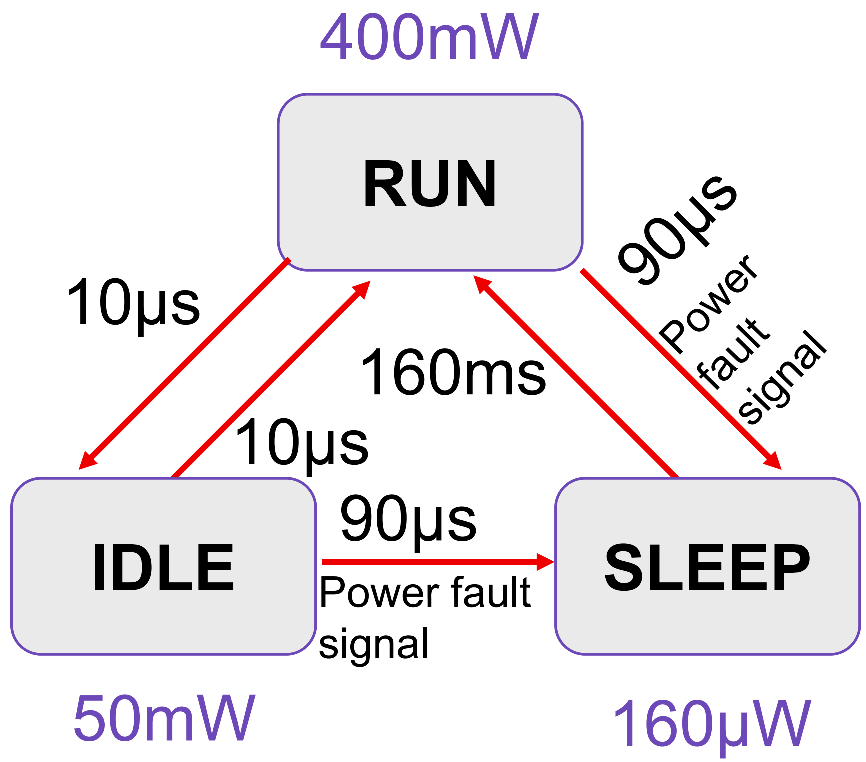
مدیریت توان پویا
Dynamic power management (DPM)

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



مبانی تغییر مقیاس پویای ولتاژ
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:

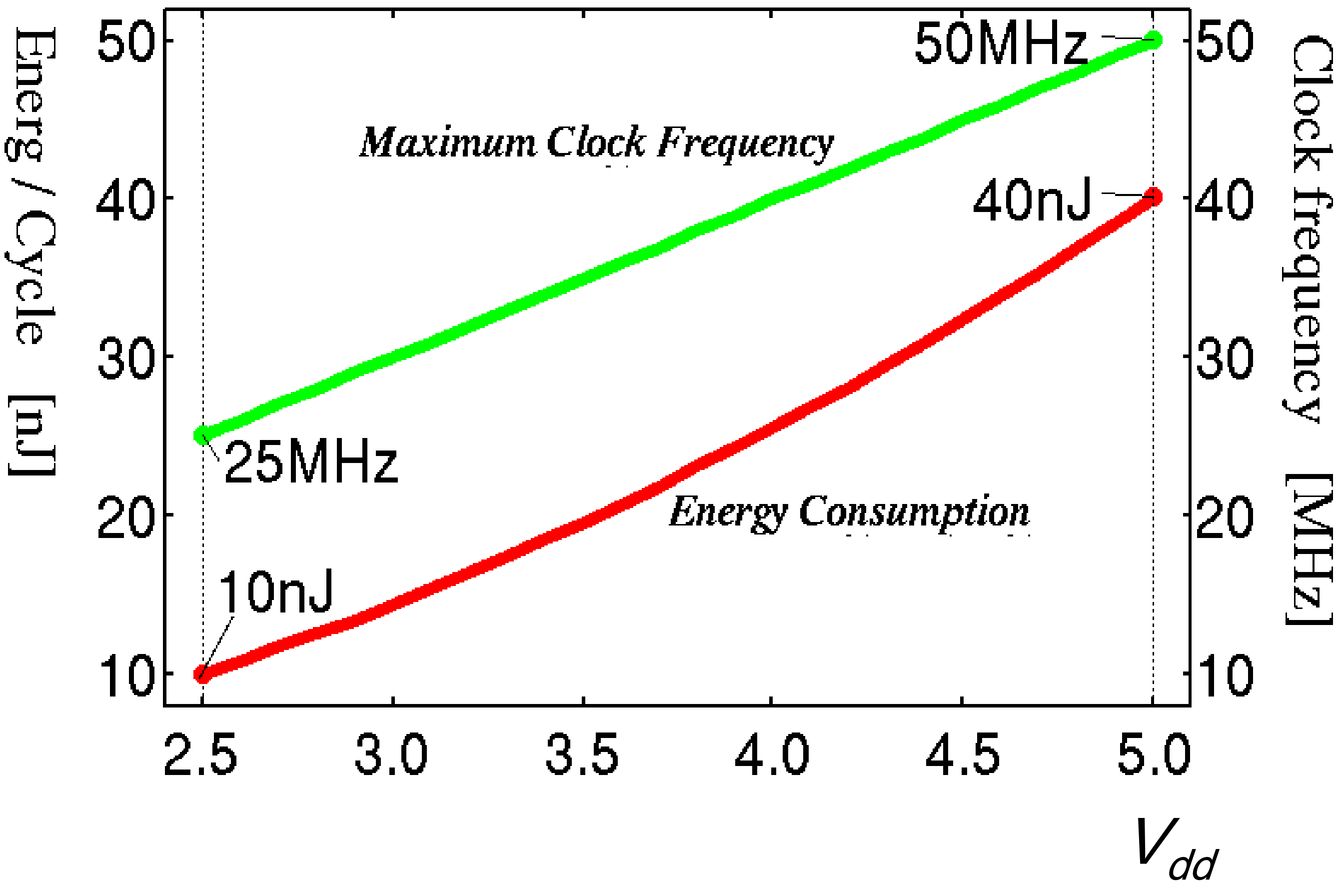
$$\tau = k \, C_L \frac{V_{dd}}{(V_{dd} - V_t)^2} \quad \text{with}$$

V_t : threshhold voltage

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

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

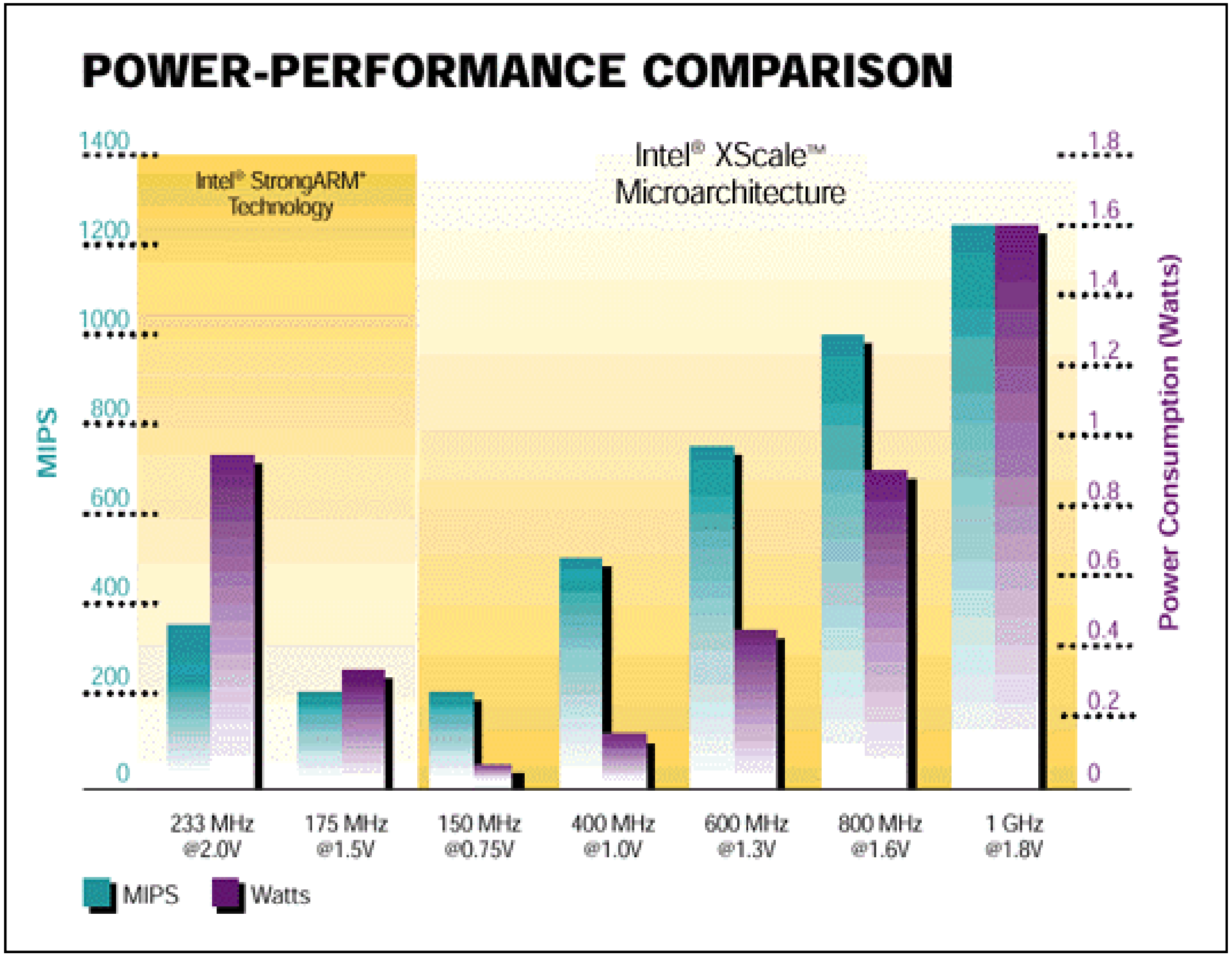
Voltage scaling: Example



[Courtesy, Yasuura, 2000]

Exploitation discussed in codesign chapter

Variable-voltage/frequency example: INTEL Xscale

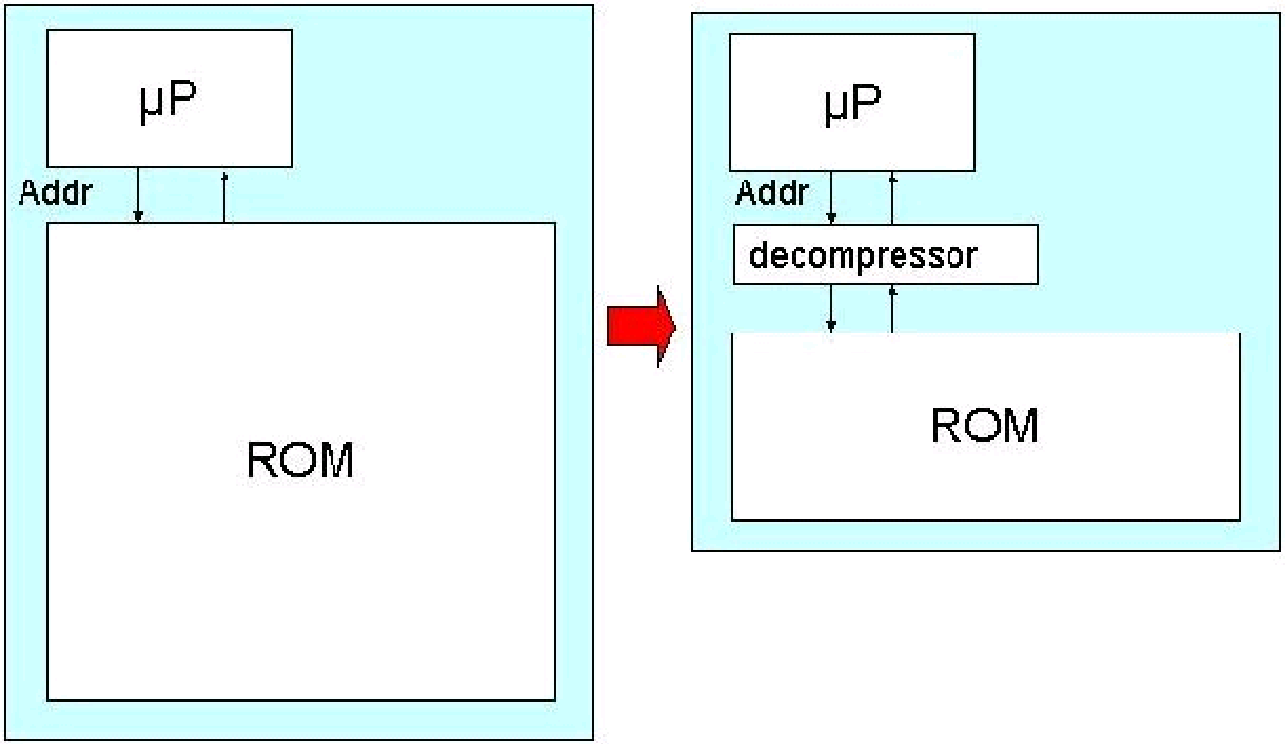


OS should schedule distribution of the energy budget.

From Intel's Web Site

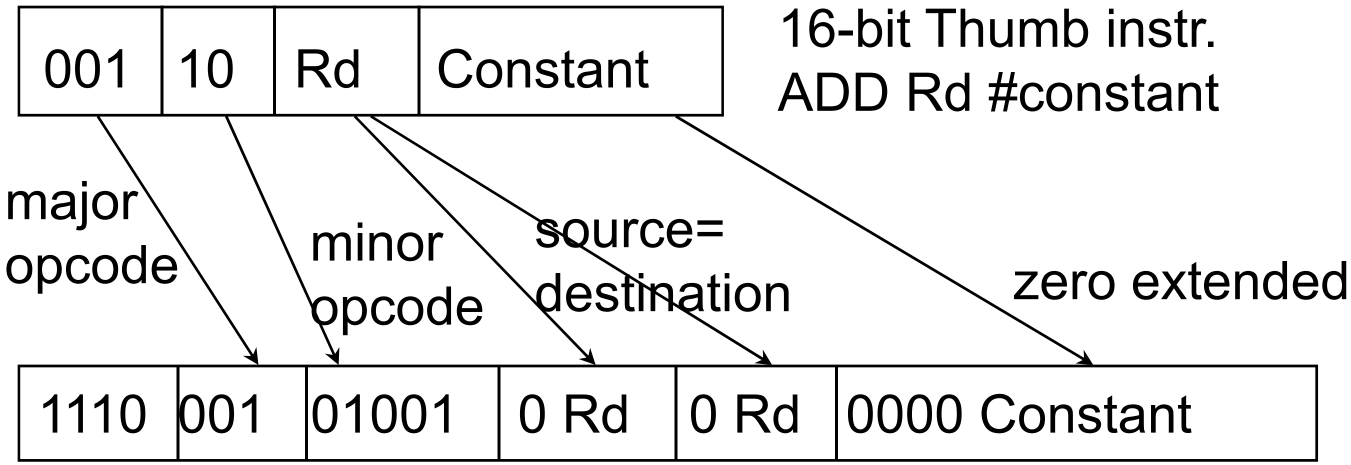
نیازمندی کلیدی ۲: کارامدی اندازه کد

- **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]

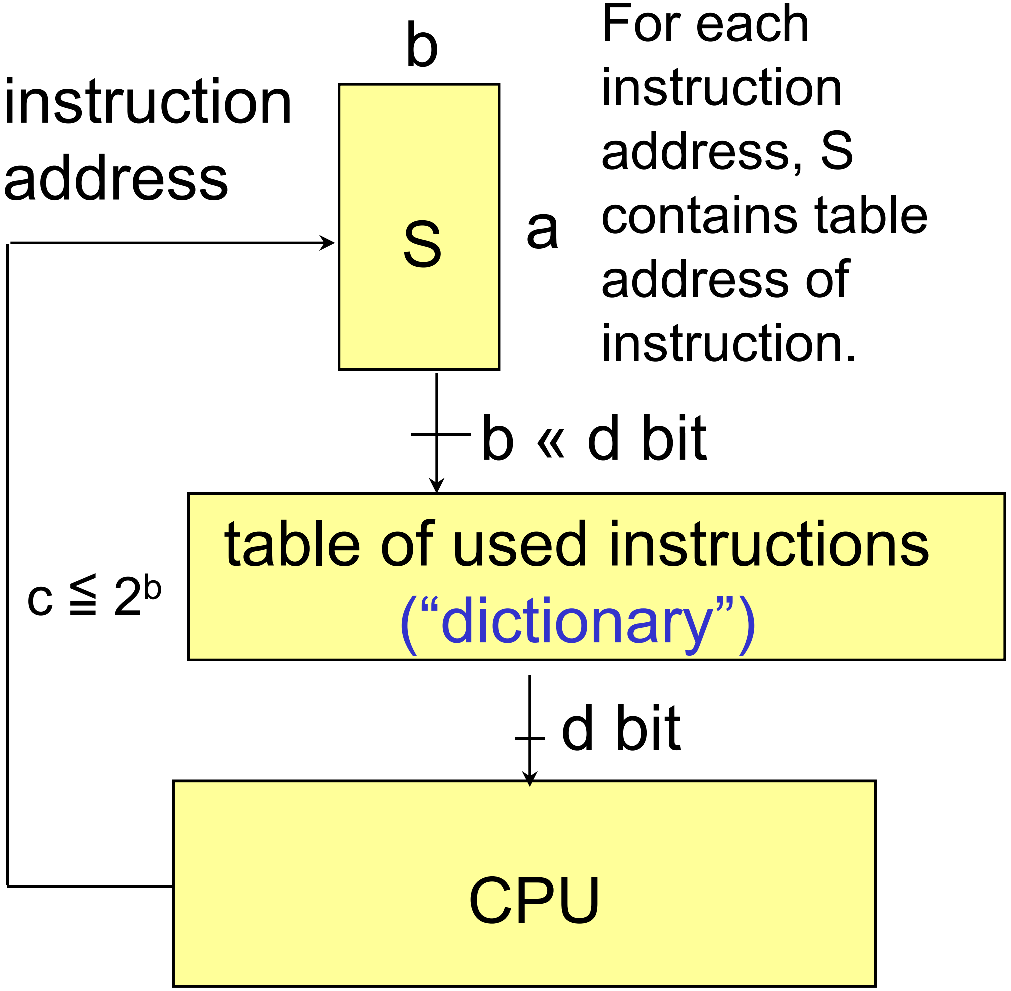
Same approach for LSI TinyRisc, …
Requires support by compiler, assembler etc.

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 $a \times d$ 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.

Instances

- Ziv-Lempel coding (☞ZIP, GZIP)
- “procedural abstraction”, “procedure exlining” (automatic generation of parameter-less procedures)
- Markov-based dictionary generation
- …

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] unfortunately has been moved ☹
- http://www-perso.iro.umontreal.ca/~latendre/codeCompression/codeCompression/node1.html

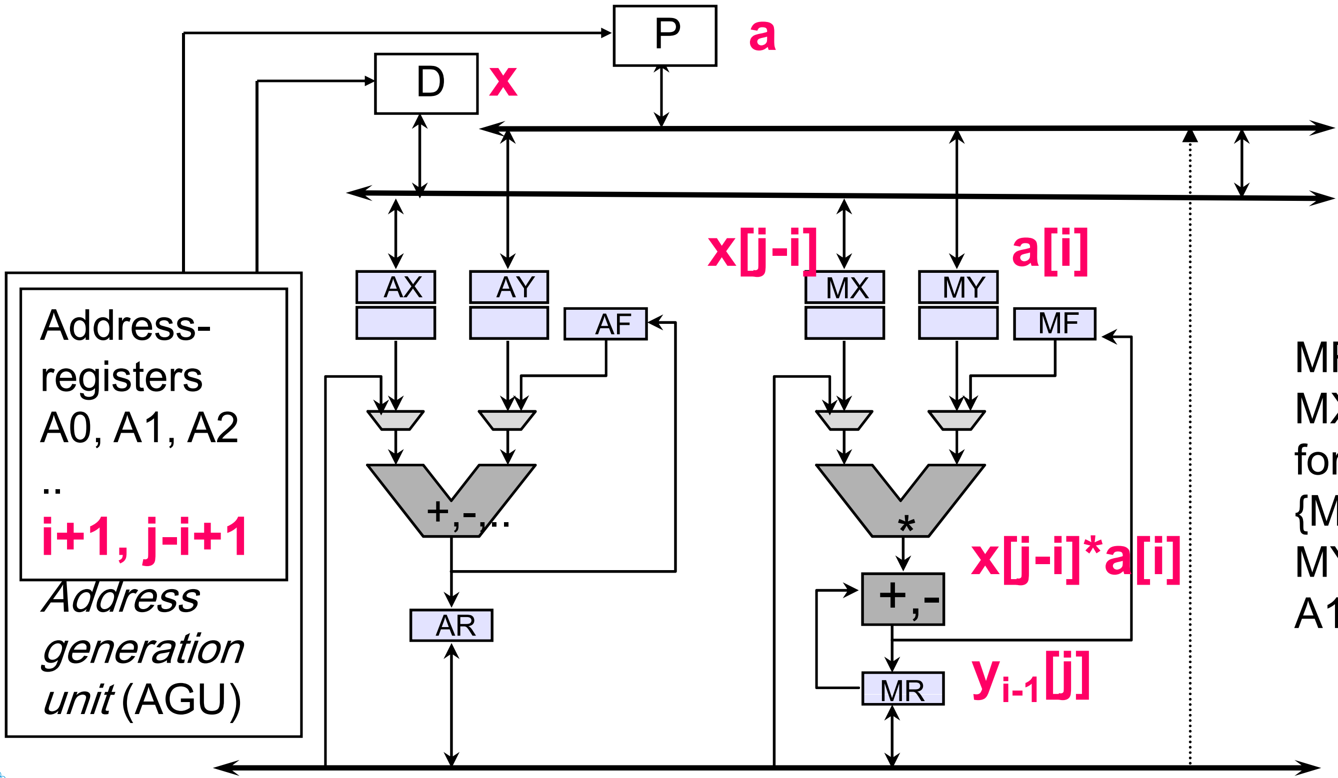
نیازمندی کلیدی ۳: کارامدی زمان اجرا

- 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; 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--}
```

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

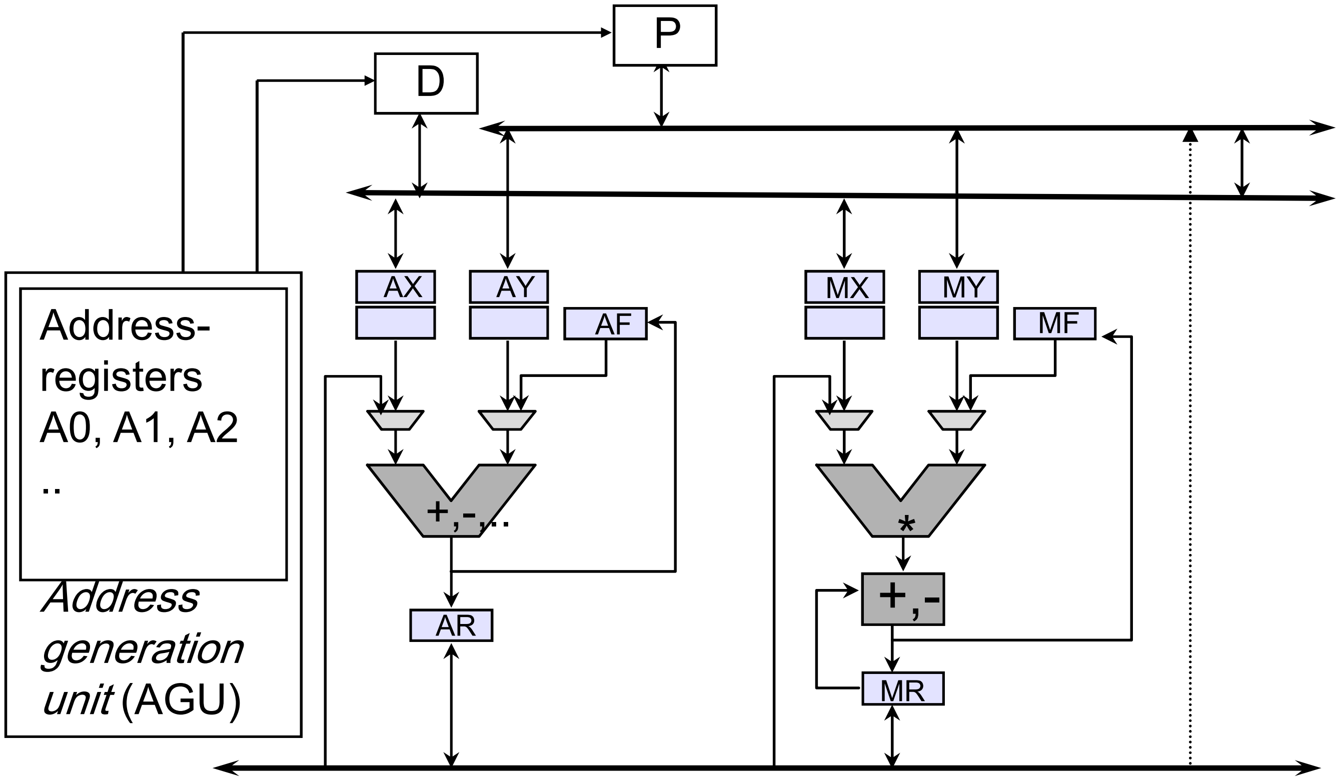
```
MR:=0; A1:=1; A2:=n-2; MX:=x[n-1]; MY:=a[0];
for ( j:=1 to n)
  {MR:=MR+MX*MY; MY:=a[A1]; MX:=x[A2]; A1++; A2--}
```

Multiply/accumulate (MAC) instruction

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

ثبات‌های ناهمگن
Heterogeneous registers

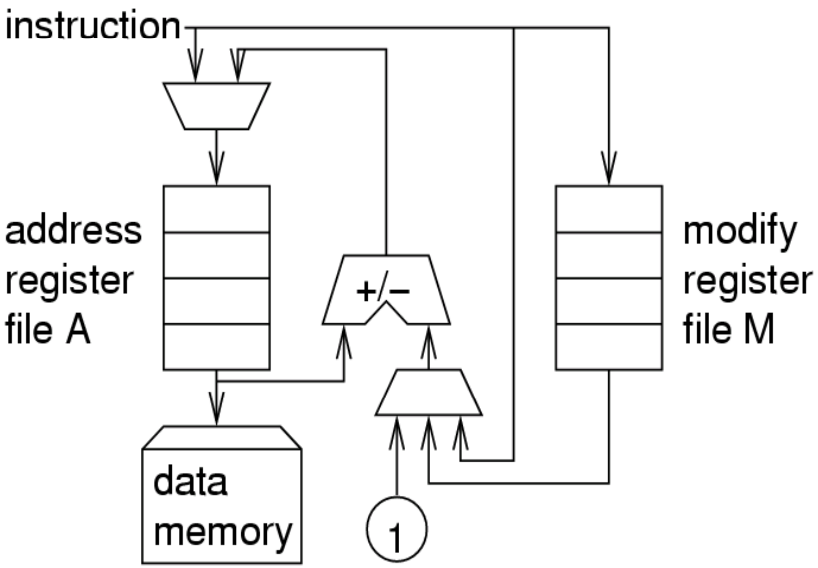
Example (ADSP 210x):



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

واحدهای تولید آدرس جداگانه
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 codesign chapter

آدرس‌دهی پیمانه‌ای
Modulo addressing

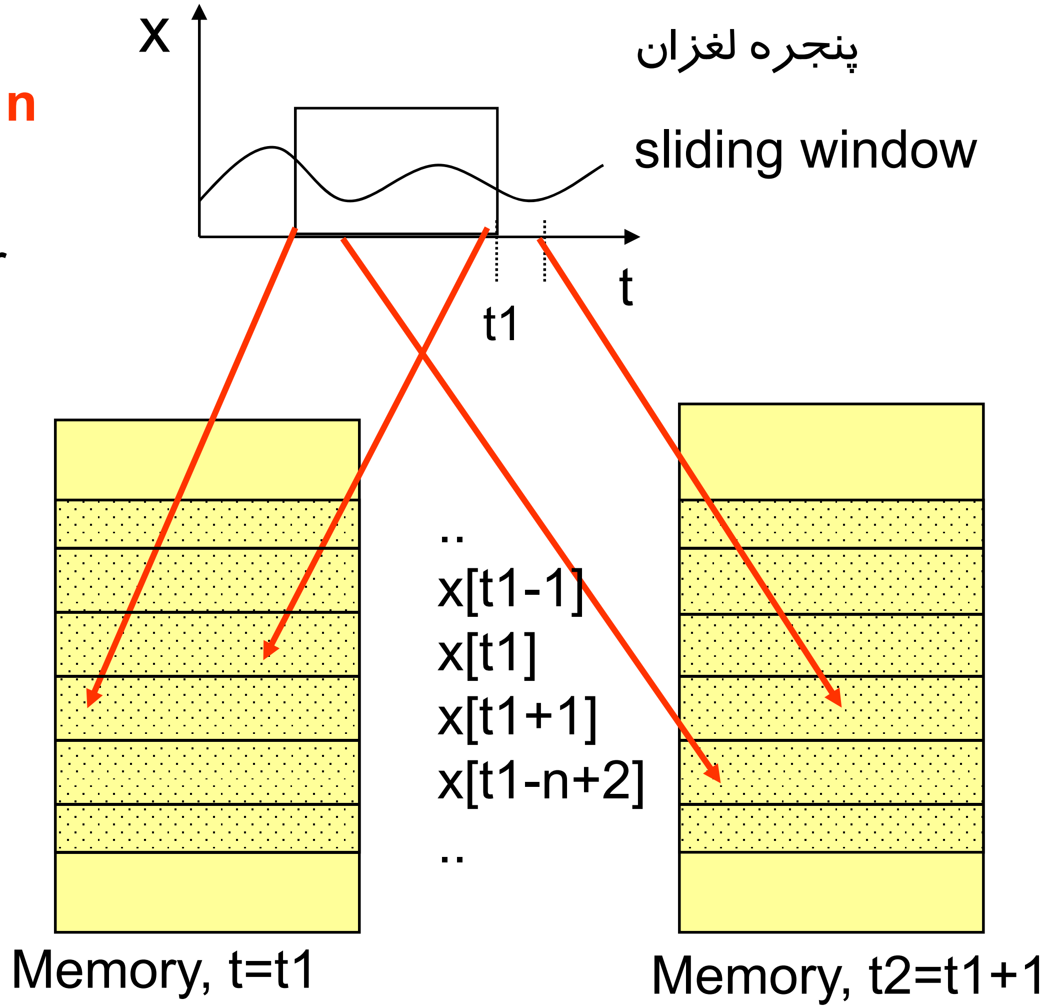
Modulo addressing:

Am++ ≡ Am:=(Am+1) **mod n**

(implements ring or circular buffer in memory)

n most recent values

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



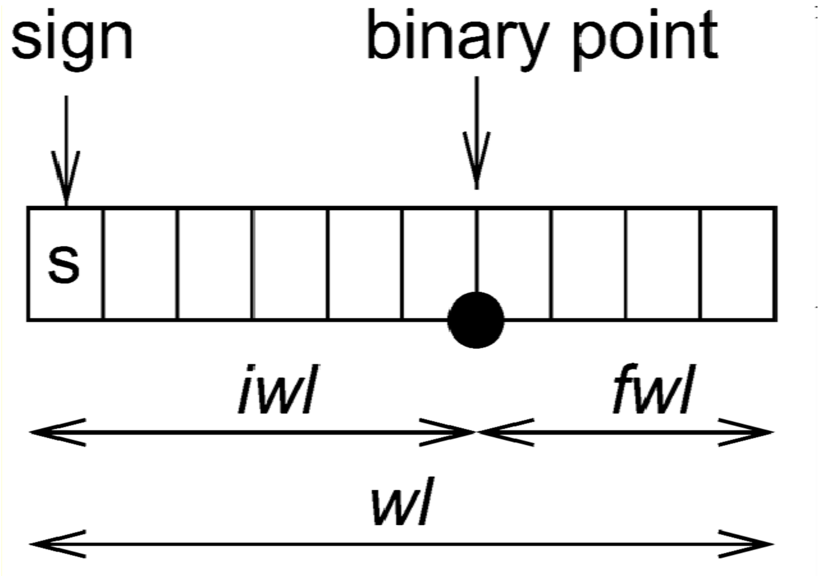
محاسبات اشباع کننده
Saturating arithmetic

- 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.

محاسبات ممیز ثابت
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 ∈ [0..1),
  with no stored exponent (bits used to increase precision).
- Appropriate for DSP/multimedia applications (well-known value ranges).

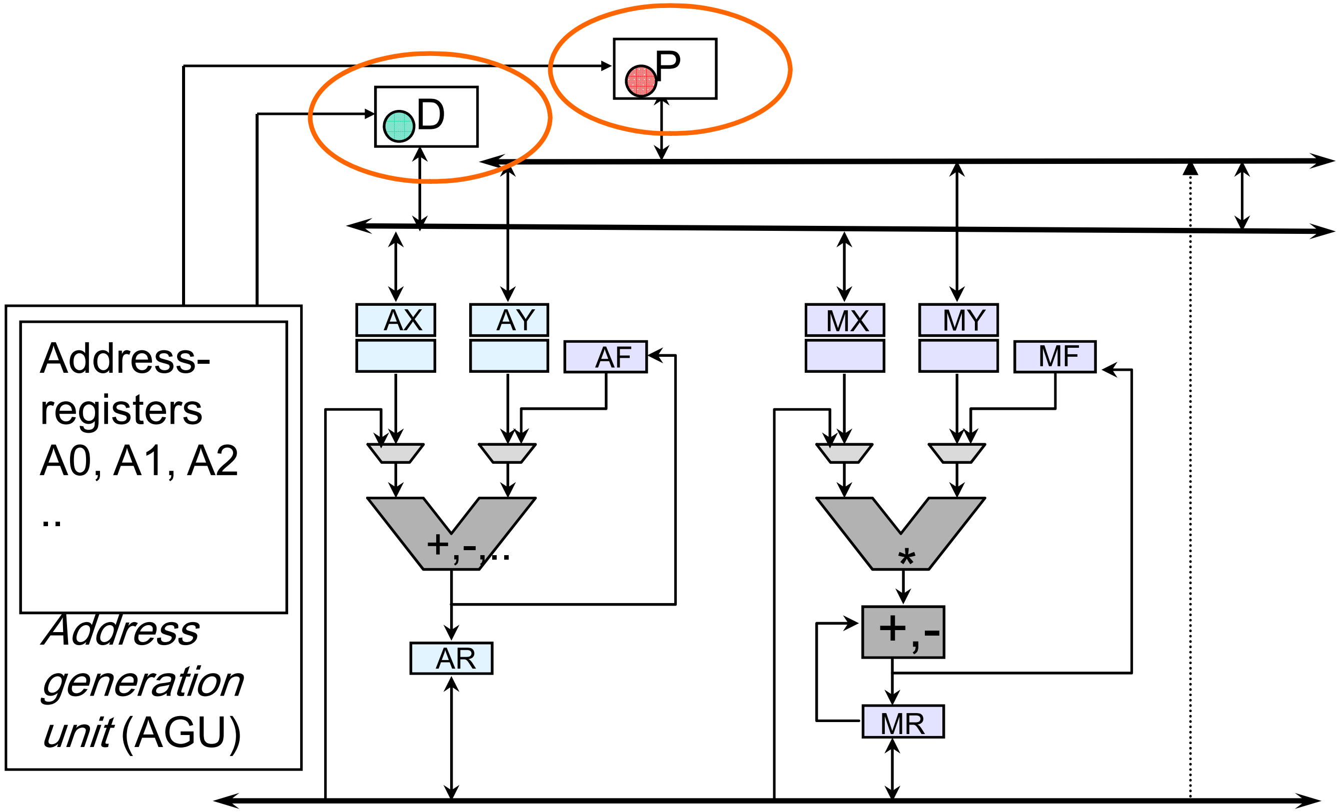
قابلیت بی‌درنگ بودن
Real-time capability

- **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 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 that are possible any time
  - Instructions that have data-dependent execution times

☞ Trying to avoid as many of these as possible.

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

بانک‌های حافظه‌ی چندگانه یا حافظه‌ها
Multiple memory banks or memories



ساده‌سازی واکشی‌های موازی

خلاصه

Processing units

- Power efficiency of target technologies
- ASICs
- Processors
  - LEGO RCX unit
  - Energy efficiency
  - Code size efficiency and code compaction
  - Run-time efficiency
  - DSP processors
    - Addressing modes, AGUs
    - Saturating and fixed point arithmetic
    - Real-time capability, multiple banks
    - Heterogeneous register files, MAC