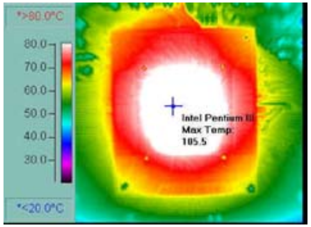
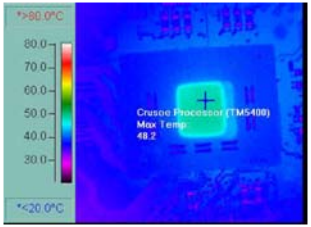
Lecture 11

Architectures for Low Power: Transmeta's Crusoe & Efficeon Processors

Processor Thermal Comparison



Intel Pentium III



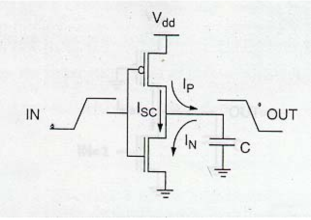
Crusoe TM5400
Processor

Motivation

- Exponential performance increase at a low cost
- However, for some application areas low power consumption is more important than performance:
  - Mobile communications
  - Mobile computing
  - Wireless Internet
  - Medical implants
  - Deep space applications
- Battery life time

Power Consumption in CMOS Circuits

- Basic classes of the consumed power
  - Static
  - Dynamic



Power Consumption in CMOS Circuits

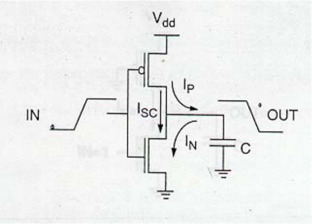
- Static Power
  - Ideally, CMOS circuits dissipate no static (DC) power since in the steady state there is no direct path from V_{dd} to ground.
  - Static component of CMOS power dissipation
    - Leakage currents
    - Subthreshold currents
    - Substrate currents
      - Little effect on overall power consumption

Power Consumption in CMOS Circuits

- Dynamic Power
  - Dynamic component of CMOS power dissipation
    - Transient switching behavior
      - With careful design for balanced input and output rise times, this component can be kept below 10-15% of the total power.
    - Capacitive switching
      - The result of charging and discharging parasitic capacitances in the circuit

Power Consumption in CMOS Circuits

- Dynamic Power (Cont.)
  - Dynamic power dissipation due to capacitive switching
    - Every time a capacitive node switches from ground to V_{dd}, an energy of CV_{dd}^2 is consumed.
    - Depends on the switching activity of the signals involved.



Power Consumption in CMOS Circuits

- Dynamic Power (Cont.)
  - Effective frequency of switching, αf
    - Activity α : The expected number of transitions / data cycle
    - Average data rate f: The clock frequency in a synchronous system
  - Average CMOS power consumption

$$P_{dyn} = \left(\frac{1}{2}CV_{dd}^2\right)\alpha f$$

    - At least 90% of the total power dissipation

Designing for Low Power: Degrees of Freedom

- 3 degrees of freedom inherent in the low-power design space

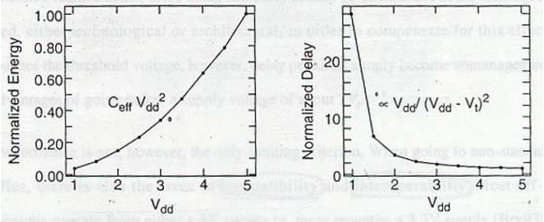
$$P_{dyn} = \left(\frac{1}{2}CV_{dd}^2\right)\alpha f$$

  - Supply voltage
  - Physical capacitance
  - Switching activity
- These parameters are not completely orthogonal and cannot be optimized independently.

Designing for Low Power: Degrees of Freedom

- Voltage
  - Quadratic relationship to power
    - The most direct and dramatic means of minimizing energy consumption
  - Factors that influence selection of a system supply voltage
    - Power
    - Performance requirements
    - Compatibility

Designing for Low Power: Degrees of Freedom



(a) Energy vs. voltage

(b) Delay vs. voltage

Designing for Low Power: Degrees of Freedom

- Physical Capacitance
  - Stems from two primary sources
    - Devices
    - Interconnect
  - Can be kept at a minimum by using small devices and short wires.
  - Device size ⇓ →Capacitance ⇓, Current drive ⇓
    - The circuit operates more slowly →prevents from lowering V_{dd}.

Designing for Low Power: Degrees of Freedom

- **Activity**
  - Determines how often switching occurs.
  - f determines the average periodicity of data arrivals, α determines how many transitions each arrival will spark.
  - Glitching
    - Spurious and unwanted transitions that occur before a node settles down to its final value.

Designing for Low Power: Approaches

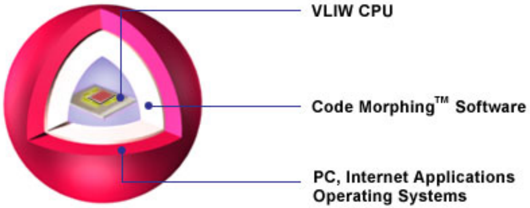
- Many of the power reduction techniques applicable at various level of abstraction follow a small number of common themes (approaches).
  - **Trading area/performance for power**
  - Avoiding waste
  - Exploiting locality

Crusoe Family of Processors from Transmeta

- Introduction
- Software
  - Code Morphing
- Hardware
  - VLIW core
- Power Management
  - LongRun
- Applications

Hardware/Software Partitioning

- Drawing the H/W and S/W line
  - Hardware: VLIW+hardware translation support
  - Software: Translates x86 code to VLIW code



Crusoe Processor
= Software+Hardware

Code Morphing software
- Dynamically translates x86 instructions into VLIW instructions
- Provides x86 compatibility
- Optimization and scheduling by software

3/4

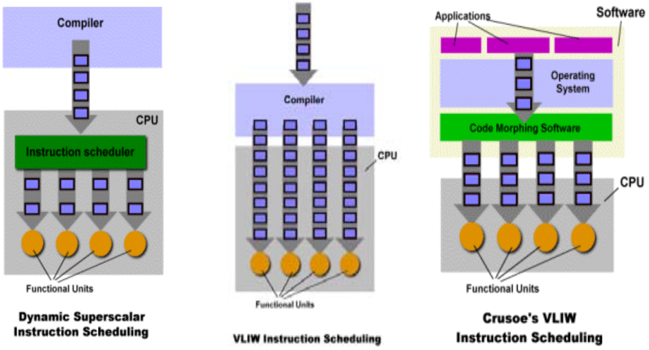
VLIW hardware
- 128 bit Very long Instruction Word Processor
- Simple and fast
- Fewer transistors

1/4

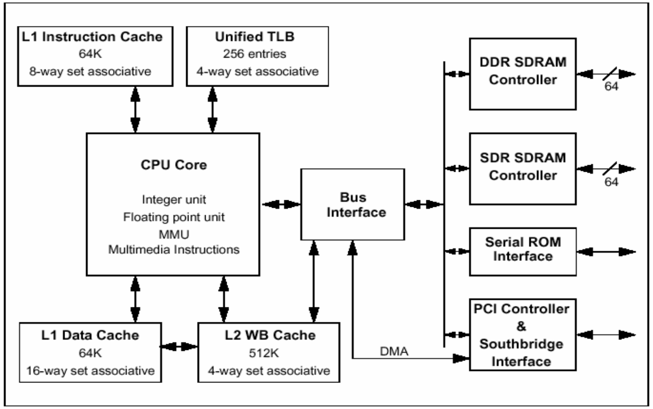
+

Low power
x86 compatibility
PC performance

Code Morphing Software

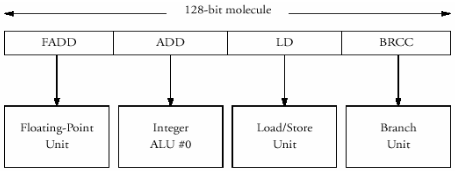


Crusoe Architecture (TM5800)

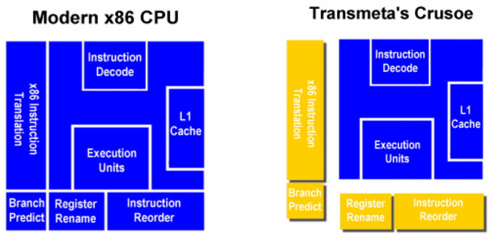


Crusoe Architecture, Cont.

- VLIW CPU : executing up to 4 operations in each cycle
  - Molecule: long instruction word (128 bits molecule)
  - All atoms within a molecule are executed in parallel
- 2 ALU, 1FP, 1 load/store, 1 branch unit
- In-order 7-stage integer/10-stage FP pipeline
- 64 integer registers, 32 FP registers



Crusoe vs. x86



- The blue stuff is silicon, and the yellow is software
- Crusoe's blue part is smaller
- All of those hardware was moved off the die and into software

Code Morphing Software

- A dynamic translation system
  - Provides the Transmeta Crusoe processor with x86 compatibility
  - Resides in ROM: first program to start executing when booting
- Benefits
  - Improvements to power consumption and performance
  - Upgrades to the software portion of a microprocessor can be rolled out independently from the chip.
  - Decoupling the hardware design from the system and application software

Code Morphing System

- Interpreter
  - Decodes and executes x86 instructions sequentially
- Dynamic binary translator
  - Select a region, produce native code and store translation in the translation cache
- Optimizer
  - Perform a number of traditional and Crusoe-specific optimizations
- Runtime system
  - Handle devices, interrupts and exceptions, power management, and garbage collection

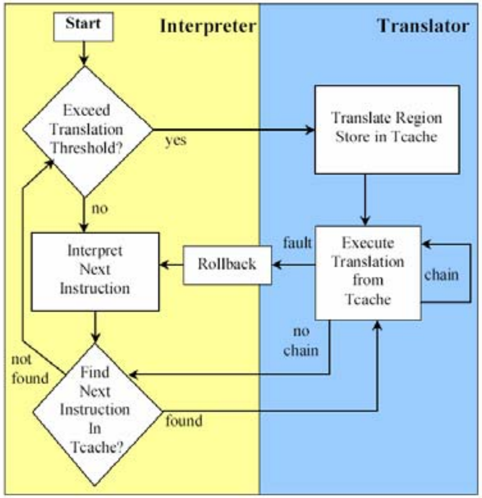

Start
Interpreter
Translator
Exceed Translation Threshold?
yes
no
Translate Region Store in Tcache
Interpret Next Instruction
Rollback
fault
Execute Translation from Tcache
chain
no chain
not found
Find Next Instruction In Tcache?
found

Figure 1: Typical CMS Control Flow

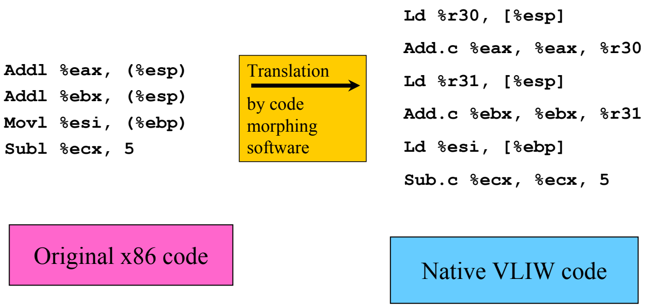
Interpretation & Translation

- Interpretation
  - Keep track of which blocks of code execute most often
    - Optimizes them accordingly
  - Keep track of which branches are most often taken
    - Annotate the code accordingly
- Translation
  - Highly optimized code
    - Takes longest to generate
    - Run faster once translated
  - Translation cache
    - Resides in a separate memory space
    - The size can be set at boot time or via the OS

Translation Process

- 1st pass (frontend)
  - Translate the x86 instructions into a simple sequences of atoms
    - Temporary register used
- 2nd pass (optimizer)
  - Well-known compiler optimization
    - Common subexpression elimination
    - Loop invariant removal
    - Dead code elimination
- 3rd pass (scheduler)
  - Reorders the optimized atoms and groups them into individual molecules
    - More effective scheduling algorithms
    - Larger window of execution

Translation Step 1



```
Addl %eax, (%esp)
Addl %ebx, (%esp)
Movl %esi, (%ebp)
Subl %ecx, 5
```

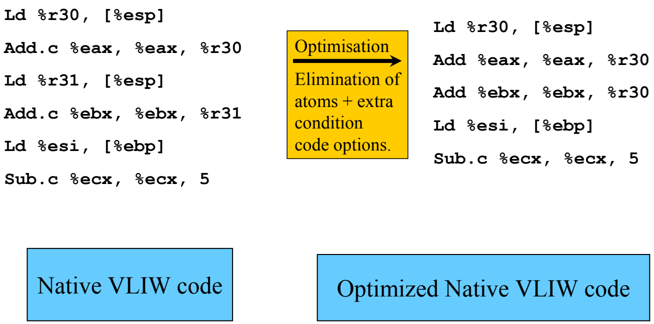
Translation → by code morphing software

```
Ld %r30, [%esp]
Add.c %eax, %eax, %r30
Ld %r31, [%esp]
Add.c %ebx, %ebx, %r31
Ld %esi, [%ebp]
Sub.c %ecx, %ecx, 5
```

Original x86 code

Native VLIW code

Translation Step 2



```
Ld %r30, [%esp]
Add.c %eax, %eax, %r30
Ld %r31, [%esp]
Add.c %ebx, %ebx, %r31
Ld %esi, [%ebp]
Sub.c %ecx, %ecx, 5
```

Optimisation → Elimination of atoms + extra condition code options.

```
Ld %r30, [%esp]
Add %eax, %eax, %r30
Add %ebx, %ebx, %r30
Ld %esi, [%ebp]
Sub.c %ecx, %ecx, 5
```

Native VLIW code

Optimized Native VLIW code

Translation Step 3

Optimized Native VLIW code

```
Ld %r30, [%esp]
Add %eax, %eax, %r30
Add %ebx, %ebx, %r30
Ld %esi, [%ebp]
Sub.c %ecx, %ecx, 5
```

↓

Scheduling -remaining atoms into molecules using a large window.

```
1. Ld %r30, [%esp]; Sub.c %ecx, %ecx, 5
2. Ld %esi, [%ebp]; Add %eax, %eax, %r30; Add %ebx, %ebx, %r30
```

Scheduled Native VLIW code

Precise Exceptions

A. addl %eax,(%esp)
B. addl %ebx,(%esp)
C. movl %esi,(%ebp)
D. subl %ecx,5

Translated to:
1. ld %r30,[%esp]; sub.c %ecx,%ecx,5
2. ld %esi,[%ebp]; add %eax,%eax,%r30; add %ebx,%ebx,%r30

- Problem
  - x86 exception in C: D should not be executed
  - In the VLIW code D is also executed

Hardware Support for Speculation and Recovery

- Two copies of each register: working copy & shadow copy
- Gated store buffer: all store operations go to a buffer

- Commit operation:
  execution successfully reaches the end of a translation
  - Copy all working registers into shadow registers
  - Write gated store buffer to the memory system

- Rollback operation:
  exceptional condition occurs inside the translation
  - Copy the shadow register values back into the working registers
  - Stores not yet committed are dropped from the gated store buffer

Reorder Loads ahead of Stores

ld %r30,[%x] // first load from location X
…
st %data,[%y] // might overwrite location X
ld %r31,[%x] // this accesses location X again
use %r31

- If the store operation does not overlap with the first load, the second load is redundant
- Translator cannot prove that load and store addressed do not overlap

Alias Hardware

- When the translator moves a load operation ahead of a store operation
  - Load => load-and-protect
    - Load and record the address and size of data loaded
  - Store => store-under-alias-mask
    - Check for protected regions
    - Raise exception

```
ldp %r30,[%x] // load from X and protect it
...
stam %data,[%y] // this store traps if it writes X
use %r30 // can use data from first load
```

Self-modifying code

- Detecting self-modifying code
  - Simply write-protect an x86 memory page.
  - If data on that protected page were later modified, fault occurred
- Discard the affected translation.
- Cost
  - Handling the fault and invalidating translations
  - Re-generating the translations

Advantages of the Code Morphing Software

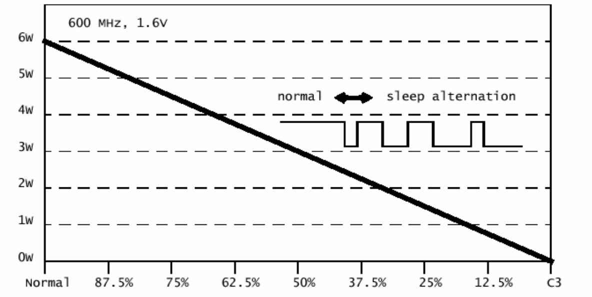
Traditional x86 Processors	Crusoe Processor with Code Morphing software
Translates each x86 instruction every time it is encountered	Translates instructions once, saving the resulted translation in a cache for re-use
Full of complex, power-hungry Transistors	Much of the processor functionality is implemented in software - less logic transistors, less power - use effective optimization/schedule algorithm - use a larger window of instruction - …

x86 Approach: ACPI Standard

- ACPI - Advanced Configuration and Power Interface
  - joint standard of Microsoft, Intel, and Toshiba
  - System level technique to reduce power
- Allows three low-power states that can be alternated
  - AutoHALT - processor executes HLT instr
    - Processor stops its internal clock
  - QuickStart - Southbridge gives processor STPCLK signal
    - Processor maintains cache coherency
  - Deep Sleep - Southbridge disables processor CLK input
    - Southbridge maintains cache coherency

Conventional Power Profile

Linear Power Savings of Conventional Power Management Solutions (Power vs. Activity Level)



LongRun Power Management

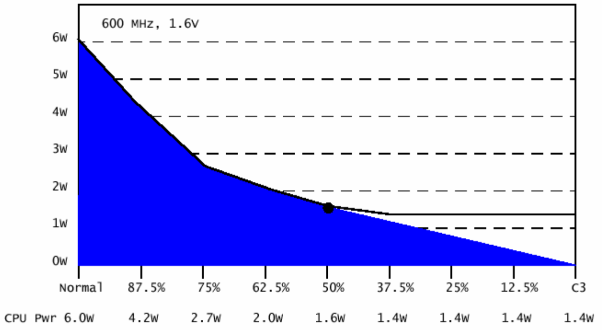
- Approach to low power consumption
  - Reduce transistor count to decrease capacitance
  - Scale voltage and frequency dynamically to give just enough performance for current workload
- LongRun
  - If no idle time detected during a workload, the frequency/voltage point is incremented
  - If idle time is detected, decrement the frequency/voltage level

LongRun Power Management, Cont.

- VLIW engine with frequency/voltage adjustments
  - Frequency changes in steps of 33 MHz
  - Voltage changes in steps of 25mV
  - Supports up to 200 frequency/voltage changes per second
- Can give cubic reductions in power consumption!

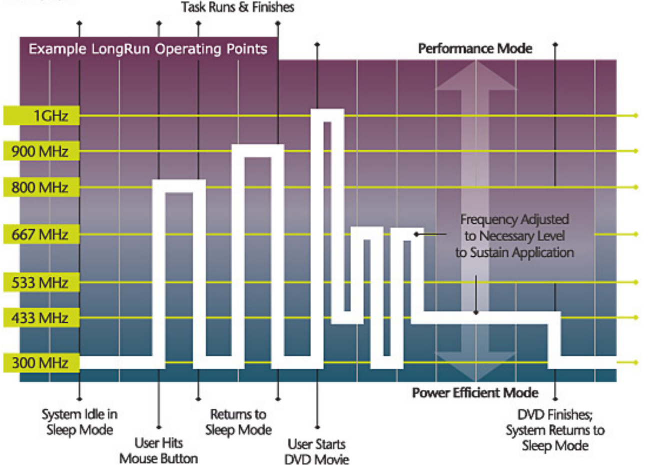
LongRun Power Profile

Figure 2: LongRun Power Profile (Power Consumption vs. Activity Level)

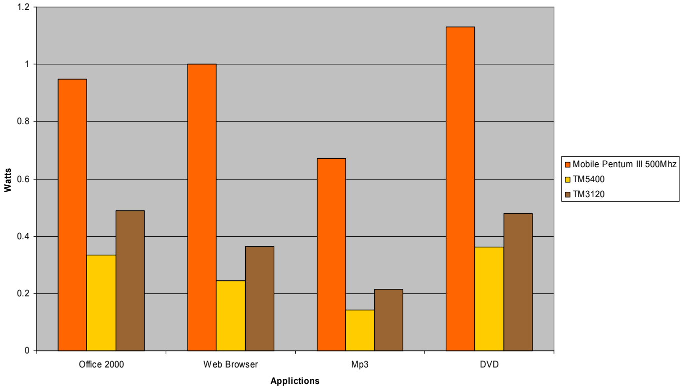


Transmeta™ LongRun™ Power Management

In Action



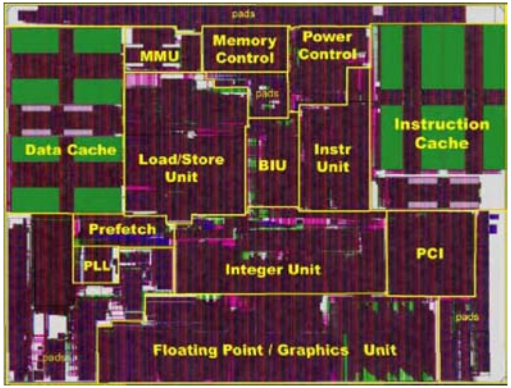
Power Consumption Comparison

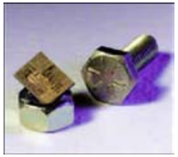


First Transmeta Chips

- TM3120
  - Originally designed for 32 bit conversions (Unix)
  - However 16 bit windows instructions translated poorly
- TM5400
  - The chip was redesigned to give better support to Windows' 16-bit applications.
  - Larger caches: improved Windows performance.
- Two chips - Two different applications.

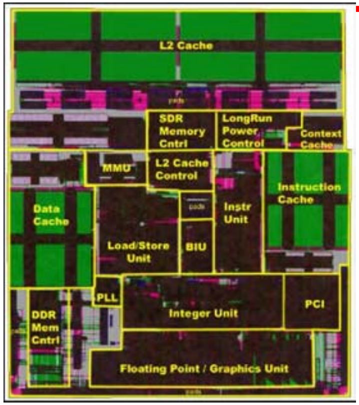
TM3120





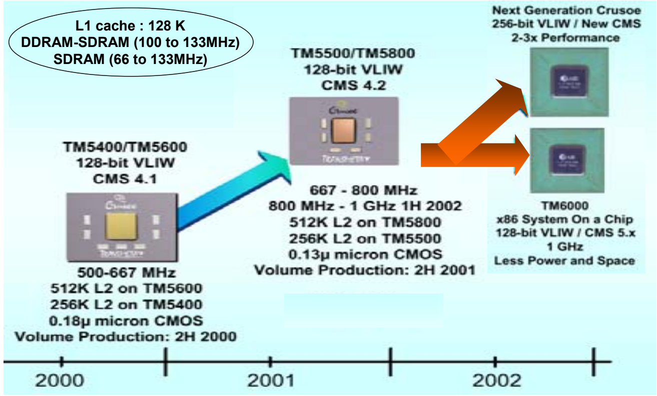
	TM3120
Frequency Range	333-400 MHz
L1 Cache	96KB
L2Cache	
Main Memory	SDRAM
Upgrade memory	
North Bridge	Integrated
Package	474 BGA
Fab Partner	IBM
Process Technology	.22u
Die Size	77mm
Sample	Now
Production	Now

TM5400



	TM5400
Frequency Range	500-700 MHz
L1 Cache	128K
L2Cache	256K
Main Memory	DDR-SDRAM
Upgrade memory	SDRAM
North Bridge	Integrated
Package	474 BGA
Fab Partner	IBM
Process Technology	.18u
Die Size	73mm
Sample	Now
Production	Mid 2000

Crusoe Processors



L1 cache : 128 K
DDRAM-SDRAM (100 to 133MHz)
SDRAM (66 to 133MHz)

TM5400/TM5600
128-bit VLIW
CMS 4.1

500-667 MHz
512K L2 on TM5600
256K L2 on TM5400
0.18µ micron CMOS
Volume Production: 2H 2000

TM5500/TM5800
128-bit VLIW
CMS 4.2

667 - 800 MHz
800 MHz - 1 GHz 1H 2002
512K L2 on TM5800
256K L2 on TM5500
0.13µ micron CMOS
Volume Production: 2H 2001

Next Generation Crusoe
256-bit VLIW / New CMS
2-3x Performance

TM6000
x86 System On a Chip
128-bit VLIW / CMS 5.x
1 GHz
Less Power and Space

2000
2001
2002

Applications

- TM3120
  - Suitable for portable and embedded systems.
  - Runs a mobile Linux kernel.
  - Capable of running Internet applications:
    - Web browsers
    - e-mail applications
    - Streaming video
- TM5400
  - Ultralite Laptops
  - Microsoft Windows compatible
  - Computer makers backing Transmeta include:
    - IBM, Fujitsu, FIC, NEC, and Hitachi

Applications, Cont.

- Currently:
  - Ultralite laptops e.g., Sharp Actius
- Processor
  - Efficeon^{TM} TM8600

Transmeta Efficeon TM8600 Processor