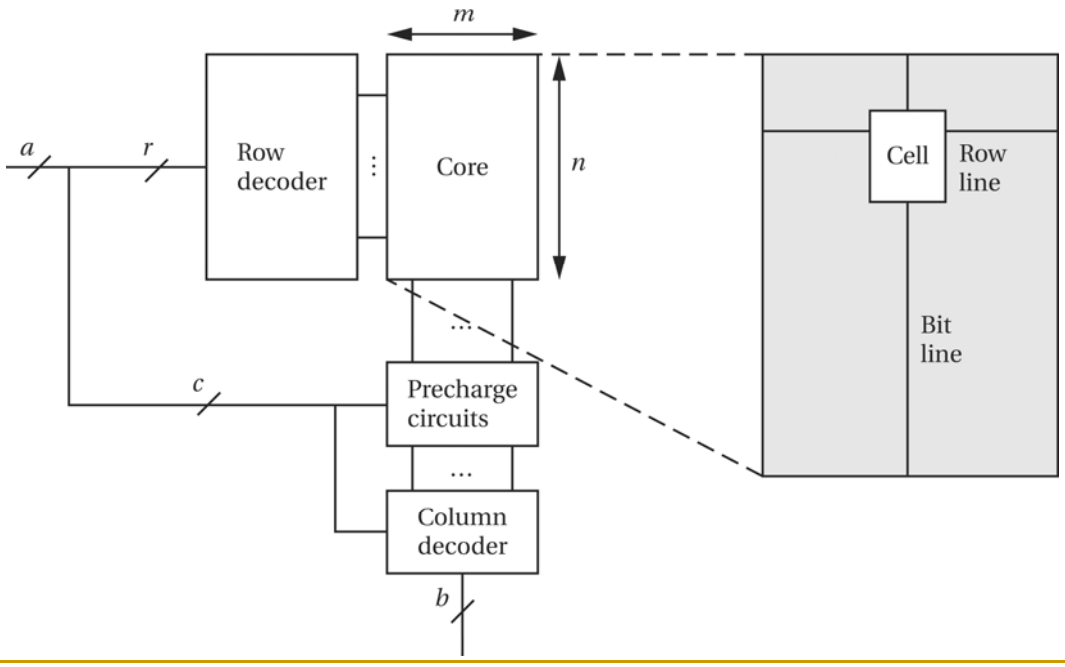
Chapter 2-2: CPUs

Soo-Ik Chae

Topics

- Memory systems.
  - Memory component models.
  - Caches and alternatives.
- Code compression.

Generic memory block



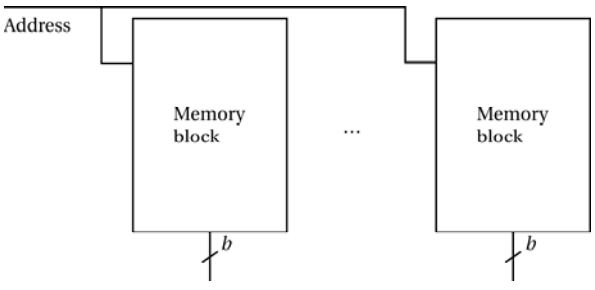
Simple memory model

- Core array is n rows x m columns.
- Total area $A = A_r + A_x + A_p + A_c$.
- Row decoder area $A_r = a_r n$.
- Core area $A_x = a_x mn$.
- Precharge circuit area $A_p = a_p m$.
- Column decoder area $A_c = a_c m$.

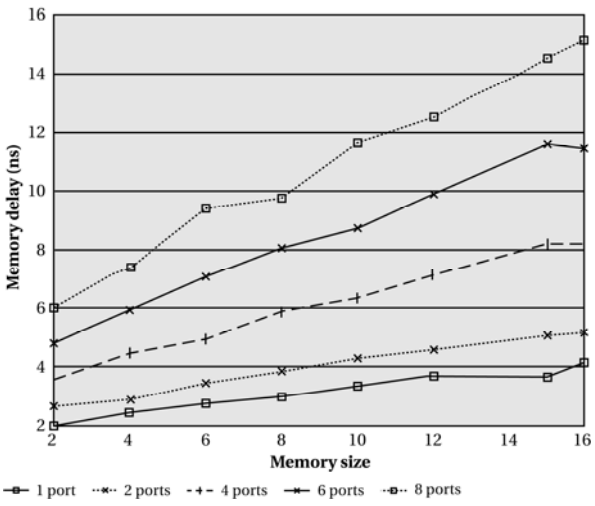
Simple energy and delay models

- $\Delta = \Delta_{setup} + \Delta_{r} + \Delta_{x} + \Delta_{bit} + \Delta_{c}$.
  - Setup delay is for the precharge circuitry
- Total energy $E = E_D + E_S$.
  - Static energy component E_S is a technology parameter.
  - Dynamic energy $E_D = E_r + E_x + E_p + E_c$.

Multiport memories



structure



Delay vs. memory size and number of ports.

Kamble and Ghose cache power model

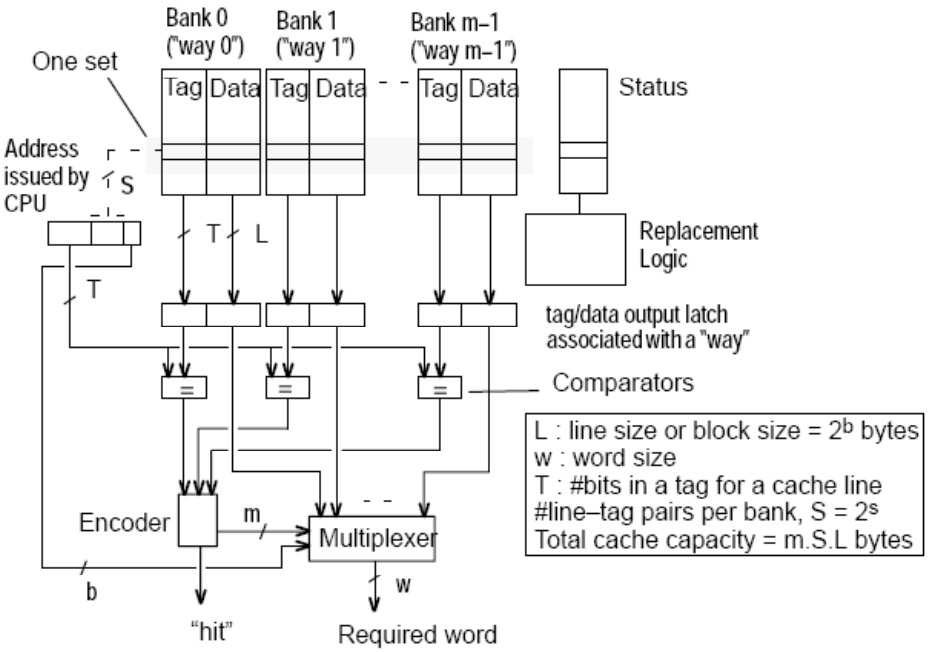


Figure 1. A m–way Set–Associative Cache

Kamble and Ghose cache power model

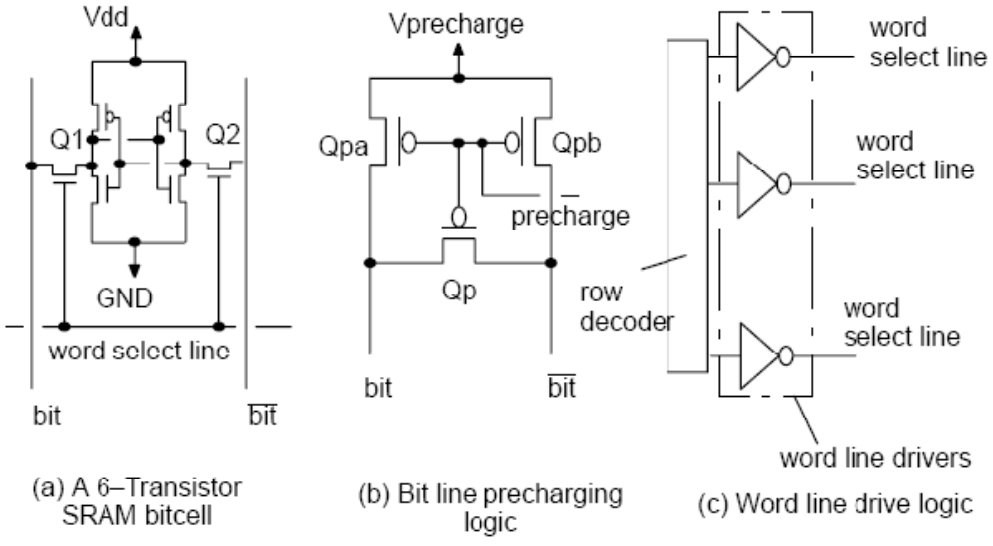


Figure 2. Some Components of a Static RAM

Kamble and Ghose cache power model

Bit–line dissipations: caused due to precharging in preparation for an access and then during the actual read or write. Based on [WiJo94] we derived the following equations:

$$C_{bit,\,pr} = N_{rows} \cdot (0.5 \cdot C_{d,Q1} + C_{bit}) \quad (2)$$

$$C_{bit,\,r/w} = N_{rows} \cdot (0.5 \cdot C_{d,Q1} + C_{bit}) + C_{d,Qp} + C_{d,Qpa} \quad (3)$$

where $C_{bit,pr}$, $C_{bit,r/w}$ are the effective load capacitance of the bit lines during precharging, and read/write to the cell. $C_{d,Qx}$ is the drain capacitance of transistor Q_x and C_{bit} is the bitwire capacitance over the extent of a single bit cell. We assume a $\frac{1}{2}V_{dd}$ voltage swing on the bit lines.

Kamble and Ghose cache power model

Word–line dissipations: caused due to assertion of the word select line by the word line drivers to perform the read or write

$$C_{wordline} = N_{columns} \cdot (2 \cdot C_{g,\,Q1} + C_{wordwire}) \quad (4)$$

where $C_{wordline}$ is capacitive load of the driver, $C_{wordwire}$ is the word select line capacitance across the extent of a bit cell.

Dissipation in output lines: caused due to driving signals on the interconnects external to the cache.

Input line dissipations: caused due to transitions on the input lines and input latches.

Kamble and Ghose cache power model

We now enumerate the energy dissipated within a m–way set associative cache, with a total data capacity of D bytes, a tag size of T bits and a line size of L bytes. Let St denote the number of status bits per block frame. These status bits are implemented as a row of $m \cdot St$ bits in a status RAM bank. The number of sets, S, is $D/(L \cdot m)$. The main components of energy dissipations are:

Capacity D = SLm [bytes]
Total Tag = STm [bits]

Kamble and Ghose cache power model

Energy dissipated in the bit lines, E_{bit}, due to precharging, readout and writes is given by:

$$N_{rows} = S \qquad N_{columns} = m \cdot (8 \cdot L + T + St)$$

$$E_{bit} = 0.5 \cdot V_{dd}^2 \cdot [N_{bit,pr} \cdot C_{bit,pr} + N_{bit,w} \cdot C_{bit,r/w} + N_{bit,r} \cdot C_{bit,r/w} + m \cdot (8 \cdot L + T + St) \cdot CA \cdot (C_{g,Qpa} + C_{g,Qpb} + C_{g,Qp})] \quad (5)$$

where $N_{bit,pr}$, $N_{bit,r}$, $N_{bit,w}$ are the total number of bit line transitions due to precharging, reads and writes, CA is the total number of cache accesses.

Kamble and Ghose cache power model

Energy dissipated in the word lines, E_{word}, including energy expended in driving the gate of the row driver

$$E_{word} = V_{dd}^2 \cdot CA \cdot m \cdot (L \cdot 8 + T + St) \cdot (2 \cdot C_{g,Q1} + C_{wordwire}) \quad (6)$$

Energy dissipated in the output lines, E_{output}, is the energy dissipated when driving interconnect lines external to the cache towards the cpu side or the memory side.

$$E_{aoutput} = 0.5 \cdot V_{dd}^2 \cdot (N_{out,a2m} \cdot C_{out,a2m} + N_{out,a2c} \cdot C_{out,a2c})$$

Kamble and Ghose cache power model

$$E_{doutput} = 0.5 \cdot V_{dd}^2 \cdot (N_{out,d2m} \cdot C_{out,d2m} + N_{out,d2c} \cdot C_{out,d2c})$$

$$E_{output} = E_{aoutput} + E_{doutput} \quad (7)$$

where $E_{aoutput}$, $E_{doutput}$ are the address and data lines dissipations, $N_{aout,a2m}$, $N_{out,d2m}$ are the number of transitions on the memory–side address and data line drivers, $C_{out,a2m}$ and $C_{out,d2m}$ are their corresponding capacitive loads. Similarly $N_{out,a2c}$, $N_{out,d2c}$ and $C_{out,a2c}$, $C_{out,d2c}$ are the corresponding terms for the cpu–side interconnect. Following [WiJo 94], the capacitive load for on–chip destinations is 0.5pF and for off–chip destinations, it is 20pF.

Shiue and Chakrabarti cache energy model

- add_bs: number of transitions on address bus per instruction.
- data_bs: number of transitions on data bus per instruction.
- word_line_size: number of memory cells on a word line.
- bit_line_size: number of memory cells on a bit line.
- E_m: Energy consumption of a main memory access.
- α, β, γ: technology parameters.

Shiue/Chakrabarti, cont'd.

$$Energy = hit\_rate * energy\_hit + miss\_rate * energy\_miss$$

$$Energy\_hit = E\_dec + E\_cell$$

$$\begin{aligned} Energy\_miss &= E\_dec + E\_cell + E\_io + E\_main \\ &= Energy\_hit + E\_io + E\_main \end{aligned}$$

$$E\_dec = \alpha * add\_bs$$

$$E\_Cell = \beta * word\_line\_size * bit\_line\_size$$

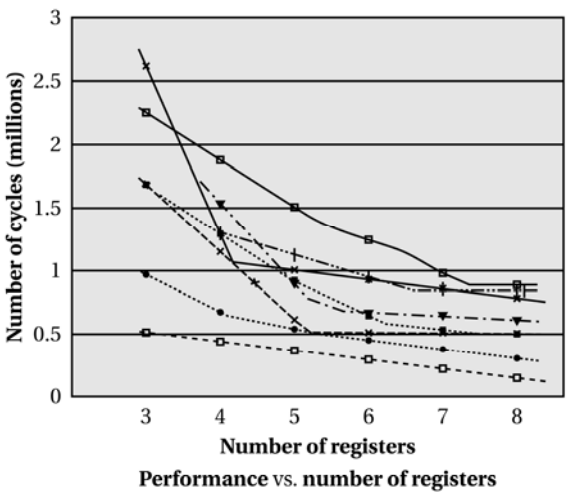
$$E\_io = \gamma * (data\_bs * cache\text{-}line\_size + add\_bs)$$

$$\begin{aligned} E\_main &= \gamma * data\_bs * cache\_line\_size \\ &\quad + Em * cache\_line\_size \end{aligned}$$

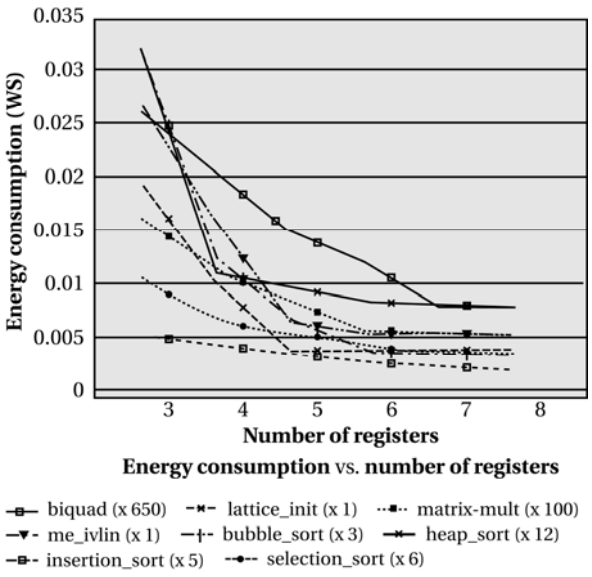
Register files

- First stage in the memory hierarchy.
- When too many values are live, some values must be spilled onto main memory and read back later.
  - Spills cost time, energy.
- Register file parameters:
  - Number of words.
  - Number of ports.

Performance and energy vs. register file size.



Performance vs. number of registers



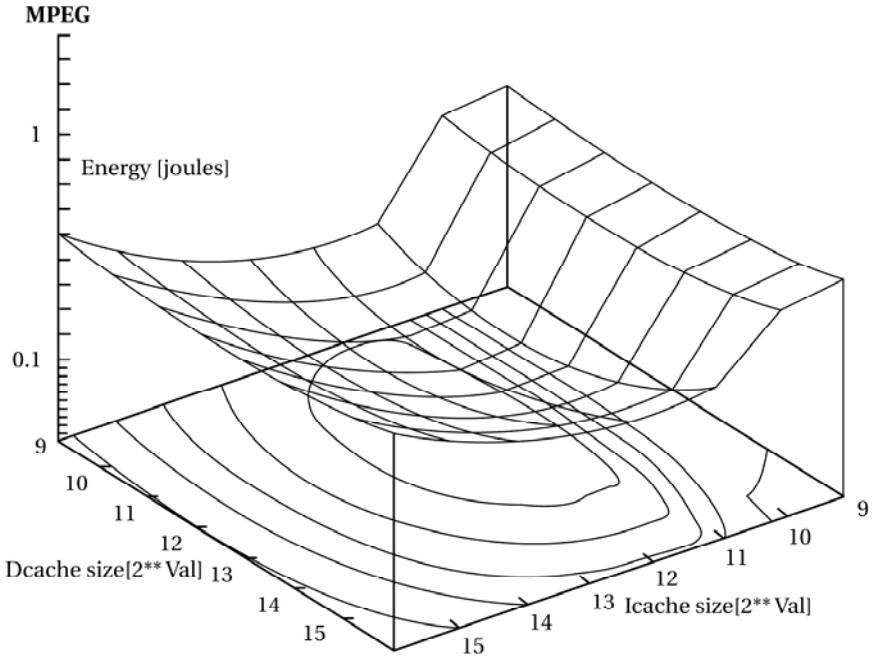
Energy consumption vs. number of registers

[Weh01] © 2001 IEEE

Caches

- Cache design has received a lot of attention in general purpose computer design
- Most of the lessons apply to embedded computer as well.
- Caches have a sweet spot: neither too small nor too large.

Cache size vs. energy



[Li98]

Cache parameters

- Cache size:
  - Larger caches hold more data, burn more energy, take area away from other functions.
- Number of sets:
  - More independent references, more locations mapped onto each line.
- Cache line length:
  - Longer lines give more prefetching bandwidth, higher energy consumption.

Wolfe/Lam classification of program behavior in caches

- Self-temporal reuse: same array element is accessed in different loop iterations.
- Self-spatial reuse: same cache line is accessed in different loop iterations.
- Group-temporal reuse: different parts of the program access the same array element.
- Group-spatial reuse: different parts of the program access the same cache line.

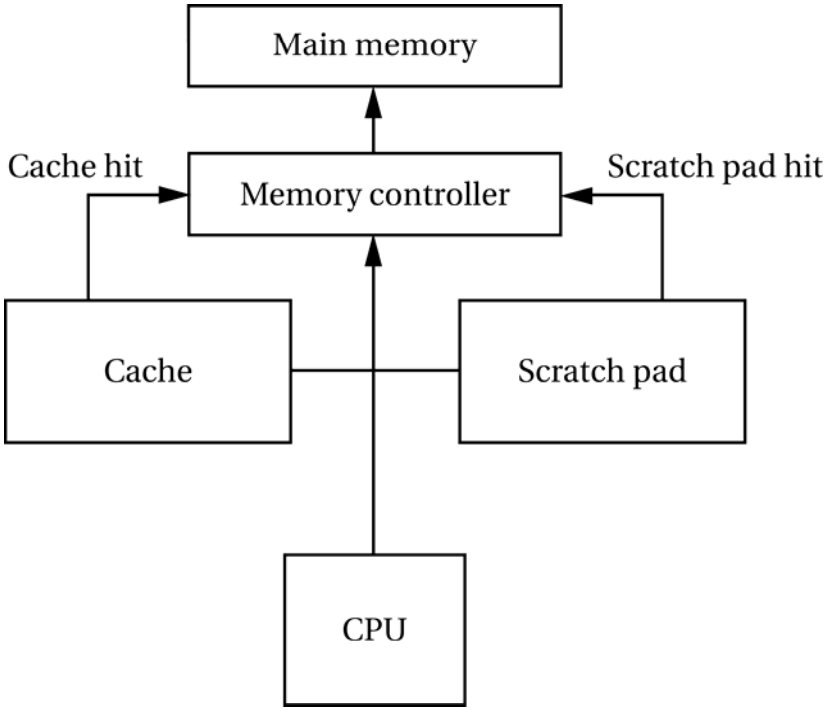
Multilevel cache optimization

- Gordon-Ross et al developed a method to optimize multilevel cache hierarchies, which adjust cache parameters in order:
  - Cache size.
  - Line size.
  - Associativity.
- Choose cache size for each level, then line size for each level, and finally associativity for each level.

Scratch pad memory

- Scratch pad is managed by software, not hardware.
  - Provides predictable access time.
  - Requires values to be allocated.
- It is a fixed part of the processor's memory space
- Use standard read/write instructions to access scratch pad.

Scratch pad memory



Code compression

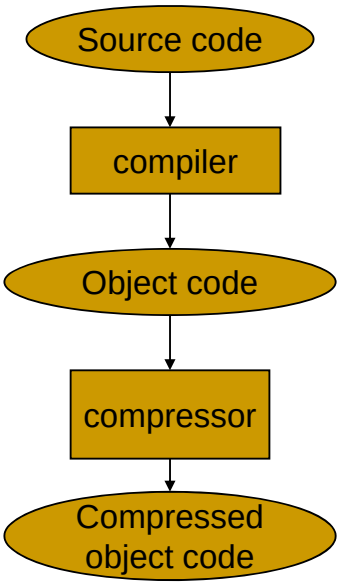
- Extreme version of instruction encoding:
  - Use variable-bit instructions.
  - Generate encodings using compression algorithms.
- Generally takes longer to decode.
- Can result in performance, energy, code size improvements.
- IBM CodePack (PowerPC) used Huffman encoding.

Terms

- Compression ratio:
  - Compressed code size/uncompressed code size * 100%.
  - Must take into account all overheads.

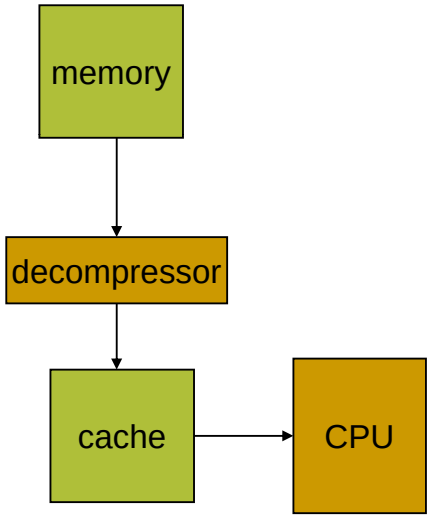
Wolfe/Chanin approach

- Object code is fed to lossless compression algorithm.
  - Wolfe/Chanin used Huffman's algorithm.
- Compressed object code becomes program image.
- Code is decompressed on-the-fly during execution.



Wolfe/Chanin execution

- Instructions are decompressed when read from main memory.
  - Data is not compressed or decompressed.
- Cache holds uncompressed instructions.
  - Longer latency for instruction fetch.
- CPU does not require significant modifications.

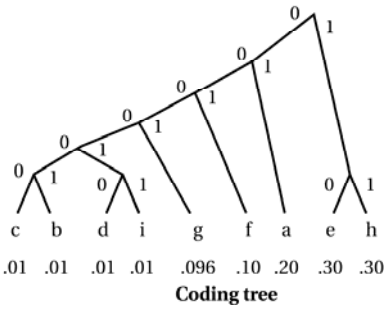


Huffman coding

- Input stream is a sequence of symbols.
- Each symbol's probability of occurrence is known.
- Construct a binary tree of probabilities from the bottom up.
  - Path from room to symbol gives code for that symbol.

a	0.20
b	0.01
c	0.01
d	0.01
e	0.30
f	0.10
g	0.096
h	0.30
i	0.01

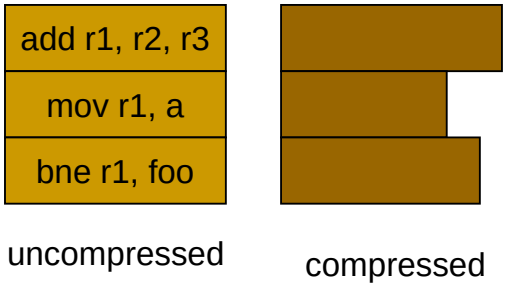
Symbols and probabilities



Coding tree

Compressed vs. uncompressed code

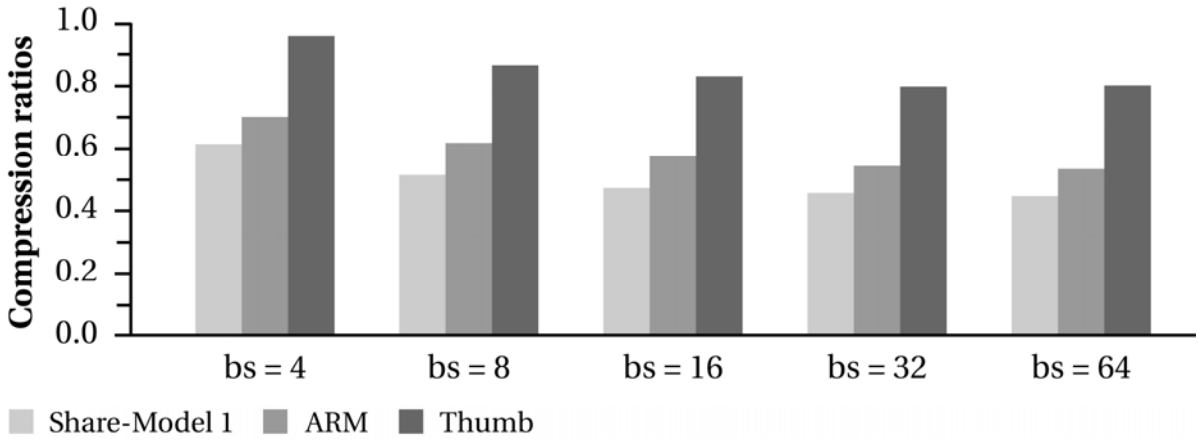
- Code must be uncompressed from many different starting points during branches.
- Code compression algorithms are designed to decode from the start of a stream.
- Compressed code is organized into blocks.
  - Uncompress at start of block.
- Unused bits between blocks constitute overhead (due to branch).



Block structure and compression

- Trade-off:
  - Compression algorithms work best on long blocks.
  - Program branching works best with short blocks.
- Labels in program move during compression.
- Two approaches:
  - Wolfe and Chanin used branch table to translate branches during execution (adds code size).
  - Lefurgy et al. patched compressed code to refer branches to compressed locations. (branch patching)

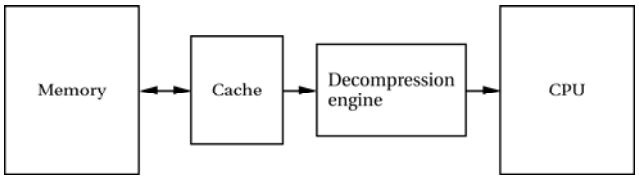
Compression ratio vs. block size



[Lek99b] © 1999 IEEE

Pre-cache compression

- Decompress as instructions come out of the cache.
- One instruction must be decompressed many times.
- Program has smaller cache footprint.



Encoding algorithms

- Data compression has developed a large number of compression algorithms.
- These algorithms were designed for different constraints:
  - Large text files.
  - No real-time or power constraints.
- Evaluate existing algorithms under the requirements of code compressions, develop new algorithms.

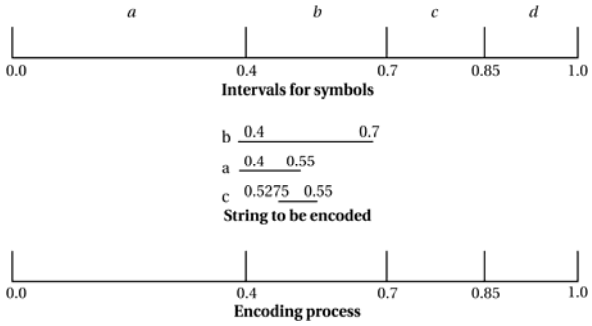
Energy savings evaluation

- Yoshida et al. used dictionary-based encoding.
- Power reduction ratio:
  - N: number of instructions in original program.
  - m: bit width of those instructions.
  - n: number of compressed instructions.
  - k: ratio of on-chip/off-chip memory power dissipation.

$$P_{f/o} = 1 - \frac{N\lceil \log n \rceil + knm}{Nm}.$$

Arithmetic coding

- Huffman coding maps symbols onto the integer number line.
- Arithmetic coding maps symbols onto the real number line.
  - Can handle arbitrarily fine distinctions in symbol probabilities.
- Table-based method allows fixed-point arithmetic to be used.



?

Code and data compression

- Unlike (non-modifiable) code, data must be compressed and decompressed dynamically.
- Can substantially reduce cache footprints.
- Requires different trade-offs.

Lempel-Ziv algorithm

- Dictionary-based method.
- Decoder builds dictionary during decompression process.
- LZW variant uses a fixed-size buffer.