Lecture Topics

- Today: Memory Management
  (Stallings, chapter 7)
- Next: continued

Announcements

- Self-Study Exercise #6
- Project #4 (due 10/12)
Memory Hierarchy

- Trade-offs among types of storage
  - Faster access time, greater cost per bit
  - Greater capacity, smaller cost per bit
  - Greater capacity, slower access speed

- Moving down the hierarchy:
  - Slower access time, decreasing cost per bit
  - Increasing capacity
  - Decreasing frequency of access by CPU
Locality of Reference

- Memory references for both instructions and data values tend to cluster over time.

- Example: once a loop is entered, there is frequent access to a small set of instructions.

- Hence: once an instruction is referenced, it is likely that the instruction (and nearby instructions) will be referenced again in the near future.

Types of Locality

- Temporal locality: same address referenced repeatedly in the near-term future
  - instructions: loops, functions
  - data: variables

- Spatial locality: nearby addresses referenced in the near-term future
  - instructions: sequential execution
  - data: arrays, similar data structures
Example: C/C++ Program

From www.cse.msu.edu/~cse410/Examples/example01

```c
int sum = 0;

int main()
{
    for( int i = 1; i <= 6; i++)
    {
        sum = sum + i;
    }
}
```

Equivalent ARM Assembly Language

```asm
.global main
.text
main:   push    {lr}
mov     r0, #1
loop:   cmp     r0, #6
        bgt     end
        ldr     r2, =sum
        ldr     r1, [r2]
        add     r1, r1, r0
        str     r1, [r2]
        add     r0, r0, #1
        b       loop
end:    pop     {lr}
mov     pc, lr
```
Equivalent ARM Machine Language

.global main
.text
0000 E52DE004 main: push {lr}
0004 E3A00001 mov r0, #1
0008 E3500006 loop: cmp r0, #6
000c CA000005 bgt end
0010 E59F2018 ldr r2, =sum
0014 E5921000 ldr r1, [r2]
0018 E0811000 add r1, r1, r0
001c E5821000 str r1, [r2]
0020 E2800001 add r0, r0, #1
0024 E4AFFFF7 b loop
0028 E49DE004 end: pop {lr}
002c E1A0F00E mov pc, lr

Execution Trace

<table>
<thead>
<tr>
<th>Time</th>
<th>PC</th>
<th>IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0010800</td>
<td>E52DE004 * main: push {lr}</td>
</tr>
<tr>
<td>1</td>
<td>0010804</td>
<td>E3A00001 mov r0, #1</td>
</tr>
<tr>
<td>2</td>
<td>0010808</td>
<td>E3500006 loop: cmp r0, #6</td>
</tr>
<tr>
<td>3</td>
<td>001080c</td>
<td>CA000005 bgt end</td>
</tr>
<tr>
<td>4</td>
<td>0010810</td>
<td>E59F2018 * ldr r2, =sum</td>
</tr>
<tr>
<td>5</td>
<td>0010814</td>
<td>E5921000 * ldr r1, [r2]</td>
</tr>
<tr>
<td>6</td>
<td>0010818</td>
<td>E0811000 add r1, r1, r0</td>
</tr>
<tr>
<td>7</td>
<td>001081c</td>
<td>E5821000 * str r1, [r2]</td>
</tr>
<tr>
<td>8</td>
<td>0010820</td>
<td>E2800001 add r0, r0, #1</td>
</tr>
<tr>
<td>9</td>
<td>0010824</td>
<td>E4AFFFF7 b loop</td>
</tr>
<tr>
<td>10</td>
<td>0010808</td>
<td>E3500006 loop: cmp r0, #6</td>
</tr>
<tr>
<td>11</td>
<td>001080c</td>
<td>CA000005 bgt end</td>
</tr>
</tbody>
</table>
Exploiting Locality

- RAM (primary storage) is slow compared to CPU registers (by a factor of about 200):
  - 0.5 ns to access registers
  - 100 ns to access RAM

- Exploit locality of reference by keeping a subset of the instructions and data values in high-speed storage (with mechanism to change the subset of instructions and data values when necessary)

Cache Memory

- Cache: fast (and thus small and expensive)
- Main memory: slow (and thus large and cheap)
- Processor first checks cache for requested word
- If not found in cache, a block of memory containing the word is moved to the cache
The Hit Ratio

- Hit ratio: fraction of accesses where item is in cache
- T1: access time for fast memory
- T2: access time for slow memory
- T2 >> T1

When hit ratio is close to 1.0, average access time is close to T1

Average Memory Access Time

Consider a two-level memory hierarchy, where M1 is faster than M2. The average memory access time can be calculated using:

\[ \text{AMAT} = H \times T1 + (1-H) \times (T1 + T2) \]
\[ = T1 + (1-H) \times T2 \]

H = hit ratio (fraction of references found in M1)
T1 = access time for M1
T2 = access time for M2
**Single Level of Cache**

**Example**

- **Processor configuration:**
  
  - cache access time is 1 clock cycle (1 ns)
  
  - cache miss penalty is 100 clock cycles

- If the requested item is in cache, then it can be accessed in one clock cycle (no delay)

- If the requested item is not in cache, then the processor has to stall until the item can be fetched from RAM
Example (continued)

- For a particular instruction sequence, the hit rate is 97%. What is the average memory access time?

\[
\text{AMAT} = \text{time for a hit} + \text{miss rate} \times \text{miss penalty} \\
= 1 \text{ clock cycle} + 0.03 \times 100 \text{ clock cycles} \\
= 4 \text{ clock cycles (or 4 ns)}
\]

Example (continued)

- Assume the hit rate is 99% instead. What is the average memory access time?

\[
\text{AMAT} = \text{time for a hit} + \text{miss rate} \times \text{miss penalty} \\
= 1 \text{ clock cycle} + 0.01 \times 100 \text{ clock cycles} \\
= 2 \text{ clock cycles (or 2 ns)}
\]
Multiple Levels of Cache

Example

- Processor configuration:
  - Level 1 cache access time is 1 clock cycle (1 ns)
  - Level 2 cache access time is 10 clock cycles
  - RAM access time is 100 clock cycles

- Check L1 cache
- If not found, check L2 cache
- If not found, fetch from RAM
Example (continued)

- For a particular instruction sequence
  - L1 cache: 90% hit rate
  - L2 cache: 80% hit rate for remaining references

- Fraction of references found at each level?
  - Level 1: 90% of all references (0.9)
  - Level 2: 80% of remaining 10% (0.08)
  - RAM: 20% of remaining 10% (0.02)

Example (continued)

- For a particular instruction sequence
  - L1 cache: 90% hit rate
  - L2 cache: 80% hit rate for remaining references

- What is the average memory access time?
  \[
  AMAT = 1 \text{ clock cycle} + 0.10 \times 10 \text{ clock cycles} \\
  + 0.02 \times 100 \text{ clock cycles} \\
  = 1 + 1 + 2 \text{ clock cycles} \\
  = 4 \text{ clock cycles (4 ns)}
  \]
Example (continued)

- For a particular instruction sequence
  - L1 cache: 95% hit rate
  - L2 cache: 80% hit rate for remaining references

- What is the average memory access time?
  \[
  AMAT = 1 \text{ clock cycle} + 0.05 \times 10 \text{ clock cycles} + 0.01 \times 100 \text{ clock cycles}
  \]
  \[
  = 1 + 0.5 + 1 \text{ clock cycles}
  \]
  \[
  = 2.5 \text{ clock cycles (2.5 ns)}
  \]

Cache and RAM Configuration

Unit of transfer between RAM and cache is one block
Each cache slot holds one block
RAM is viewed as being divided into fixed-size blocks
Read Operation

Load instruction: copy data from RAM to CPU

Check cache first – if desired item is already present in the cache, simply copy item from cache to CPU

If desired item is not already present in the cache, copy a block (item and its neighbors) from RAM to the cache and copy the item to the CPU
Write Operation

Store instruction: copy data from CPU to RAM

Check cache first – if desired item is already present in the cache, simply copy item from CPU to cache

If desired item is not already present in the cache, copy item (and its neighbors) from RAM to the cache and copy the item from the CPU

Write Policies

After a store instruction, cache and RAM are inconsistent: contents of block in cache and RAM are different

Two strategies:
• Write through
• Write back
Write Policies

- Write through: whenever a cache block is changed, the block is written (copied) to RAM
- Write back: cache block is only written (copied) to RAM when the cache line is evicted (replaced)
  - multiple store instructions can occur before block has to be written to RAM
  - modified bit used to indicate that block has been changed (and must be written to RAM)

Cache Organizations

(a) Direct mapping

(b) Associative mapping
Direct Mapping

Mapping function:

\[ i = j \mod m \]

where

- \( i \) = cache line number
- \( j \) = main memory block number
- \( m \) = number of lines in the cache

Each block maps to exactly one cache line
Example Configuration

- Cache: 64 KB
- RAM: 16 MB (24-bit addresses)
- Block size: 4 bytes
- Cache is organized as $2^{14}$ lines, where each line holds 4 bytes
- RAM is viewed as 4M blocks of 4 bytes each

Example

- Address (24 bits) viewed as three fields:
  - Word: 2 bits to identify byte within word
  - Line: 14 bits to identify cache line
  - Tag: 8 bits (remaining bits)
Example (continued)

Address: 16339C

Address in binary:

000101100011001110011100

Tag: 00010110 (16)
Line: 00110011100111 (0CE7)
Word: 00 (0)

Example (continued)

Cache line  Addresses of RAM blocks

0       000000, 010000, ..., FF0000
1       000004, 010004, ..., FF0004
2       000008, 010008, ..., FF0008
       .
       .
2^{14} - 1  00FFFC, 01FFFC, ..., FFFFFC
Direct-Mapped Cache
Fully Associative Mapping

With fully associative mapping, any block can be placed in any cache line.

More flexible than direct mapping, but requires more circuitry to find a particular entry: comparator for every cache line.
Fully Associative Cache

Note: Memory address values are in binary representation, other values are in hexadecimal.