Memory systems

- Memory technology
- Memory hierarchy
- Virtual memory

1

Memory technology

- DRAM – Dynamic Random Access Memory
  - bits are represented by an electric charge in a small capacitor
  - charge leaks away, need to be refreshed at regular intervals
  - reading the memory also discarges the capacitors
- DRAM has better price/performance than SRAM
  - also higher densities, need less power and dissipate less heat
- SRAM – Static Random Access Memory
  - based on gates, a bit is stored in 4–6 transistors
  - no refreshing, retain data as long as they have power
- SRAM provides higher speed
  - used for high-performance memories
  - cache, video memory, ...

2

Access time and cycle time

- Memory access time is the time it takes read or write a memory location
- Memory cycle time is the minimum time between two successive memory references
  - can not do repeated accesses immediately after each other
  - have to refresh the memory after an access
- *Example*:
  - 50 ns access time
  - 100 ns cycle time

3

Memory banks and interleaving

- The memory is organized as a number of banks
  - each bank consists of a separate memory device
- Interleaving
  - consecutive memory accesses address different banks
  - when one bank is refreshed, another bank can be accesed
  - can overlap accesses and refreshing
- Gives a continous flow of data from memory
- Example:
  - 2-way interleaved memory

0	1
2	3
4	5
6	7
8	9
10	11
12	13
14	15

4

Dynamic RAM technology

- Fast page mode DRAM
  - improves access to memory in sequentially located addresses (cache lines)
  - the entire address does not have to be transmitted to the memory for each access, only the least significant bits
- EDO RAM (Extended Data OUT RAM)
  - very similar to fast page mode RAM
- SDRAM (Synchronous DRAM)
  - CPU and memory is synchronized by an external clock
  - consecutive data is output synchronously on a clock pulse
  - memory chips are divided into two independent cell banks, interleaving
  - PC66, PC100, PC133 SDRAM, etc.
  - 133 MHz * 64 bits / 8 bits = 1064 MB/s peak bandwidth
  - typical efficiency approx. 75 % = 800 MB/s

5

Dynamic RAM technology (cont.)

- DDR SDRAM (Double Data Rate SDRAM)
  - memory architecture chosen by AMD
  - synchronous DRAM
  - the memory chips perform accesses on both the rising and falling edges of the clock
  - a memory with a 133 MHz clock operates effectively at 266 MHz
  - 64-bit data bus
  - 133 MHz clock cycle * 2 clocks/cycle * 64 bits / 8 bits = 2128 MB/s peak bandwidth
  - typical efficiency approx. 65 % = 1380 MB/s
  - 184 pin SIMMs

6

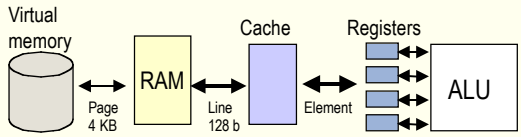
Dynamic RAM technology (cont.)

- Direct RAMBUS
  - proprietary technology of Rambus Inc., memory architecture chosen by Intel
  - new, fast DRAM architecture, 400 MHz
  - operates on both rising and falling edge of clock cycle
  - transfers data ovar a narrow 16-bit bus (Direct Rambus Channel)
  - multiple memory banks
  - use pipelining technology to send four 16-bit packets at a time (64-bit memory accesses)
  - 400 MHz * 2 clocks/cycle * 16 bits / b bits = 1600 MB/s
  - typical efficiency approx. 65 % = 1360 MB/s
- RIMMs
  - similar as DIMMs but different pin count (184 vs. 168)
  - covered by an aluminium heat spreader

7

Memory hierarchy

- Hierarchical memory organisation
  - registers
  - cache memory
  - main memory
  - disk memory



- From small, fast and expensive to large, slow and cheap
- *Example*: memory access times on a 500 MHz 21164 Alpha
  - register 2 ns
  - L1 (on-chip) 4 ns
  - L2 (on-chip) 5 ns
  - L3 (off-chip) 30 ns
  - memory 220 ns

8

Registers

- Small, very fast memory storage located close to the ALU
- Implemented by static RAM
  - operates at the same speed as instruction execution
- IA-32 ISA defines 8 general purpose 32-bit registers
  - + special purpose registers: EIP,EFLAGS, 6 segment registers
  - + 8 80 bit floating-point registers and 6 special-purpose registers
  - 8 64-bit MMX registers, aliased to FP registers
- GPR are used by the processor for operand evaluation
  - stores intermediate values in expression evaluation
  - Example:
    `x = G*2.41 + A/M -W*M`
- Optimizing compilers make efficient use of registers for expression evaluation

9

Caches

- Small, fast memory located between the processor and main memory
  - implemented by static RAM
  - store a subset of the memory
- Separate caches for instructions and data
  - Harvard memory architecture
  - can simultaneously fetch instructions and operands
- Data in a lower memory level is also stored in the higher level
- Strategies to maintain coherence between cache and memory:
  - write-through: data is immediately written back to memory when it is updated
  - write-back: data is written to memory when a modified cache line is replaced in cache

10

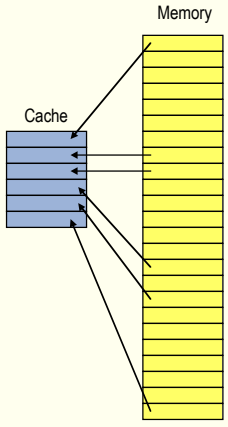
Cache lines

- The unit of data transferred between RAM and cache is called a cache line
  - consists of *N* consecutive memory locations
- When we access a memory location, a consecutive memory block is copied to the cache
  - a cache replacement policy defines how old data in cache is replaced with new data
  - tries to keep frequently used data in the cache
  - Typical cache line sizes range from 128 bits to 512 bits
- For each memory access, the computer first checks if the cache line containing this memory location is in cache
  - if not (on a cache miss) the line is brought in
  - has to decide which old cache line to throw out – LRU algorithm

11

Cache organization

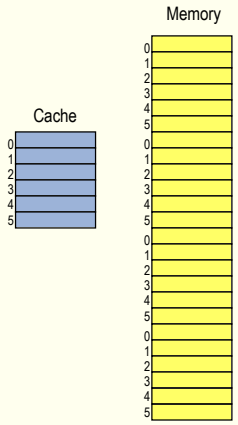
- A cache mapping defines how memory locations are placed into caches
  - mapping of addresses to cache lines
- Each cache line records the memory addresses it represents
  - called a tag
  - used to find out which memory addesses are stored in a cache line
- Cache is much smaller than RAM
  - two memory blocks can be mapped to the same cache line
- Think of memory as being divided into blocks of the size of a cache line



12

Direct mapped caches

- A memory block can be placed in exactly one cache line
- The mapping is
  - (block address) MOD (nr. of lines in cache)
- Easy to find out if a memory address is in cache or not
  - check the tag in the cache line where it is supposed to be



13

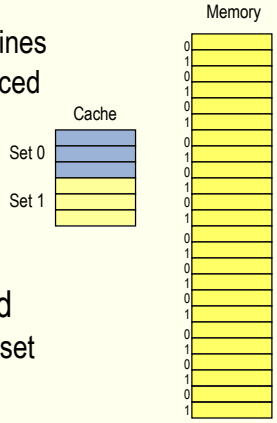
Fully associative cache

- A memory block can be placed in any cache line
- Can not calculate in which cache line a memory block should be placed
  - have to search through all cache lines to find the location containing the tag we are looking for
- Associative memory
  - search through all cache lines simultaneously for a matching tag
- Associative caches are small and expensive

14

Set associative cache

- A memory block can be placed in a restricted set of cache lines
  - the block is first mapped onto a set of cache lines
  - then it is decedied into which of these it is placed
- The mapping is
  - (block address) MOD (nr. of sets)
- If there are *N* sets, the cache placement is called *N*-way associative
- Can compute in which set a block is placed
  - only have to do associative search in a small set



15

Cache misses

- Assume a L1 cache access time of 5 ns, L2 access time of 10 ns and memory access time of 200 ns
  - if we have a 80% L1 hit rate, 15% L2 hit rate and 5% memory references the average memory access time is 0.8*5 + 0.15*10 + 0.05*200 = 15.5 ns
- Caches are based on the principle of locality
  - spatial locality – we access data located near each other (sequential access)
  - temporal locality – we do repeated accesses to the same data
- Three different reasons for cache misses
  - compulsory
  - capacity
  - conflict

16

Compulsory cache misses

- Cold start misses or first reference misses
  - the first access to a block of memory always causes a cache miss when the line is brought in to the cache
- Can increase the cache line size
  - increases cache miss penalty
  - increases conflict misses, because the cache contains fewer lines
- Can use prefetching
  - bring in the next contigous cache line at the same time
  - some processors also have a prefetch instruction, which the compiler can insert into the code
  - works for contiguous memory accesses, not for random access patterns

17

Capacity cache misses

- The cache can not hold all of the memory referenced in the program
  - capacity misses occur when some cache lines are replaced because the cache is full and later need to be brought in again
- Capacity misses can be overcome by increasing the cache size
- Can also modify the data structures and algorithm to improve spatial and temporal locality
  - compiler optimization
  - high-level code optimization techniques

18

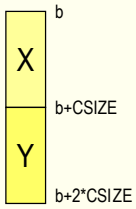
Conflict cache misses

- In direct mapped and set associative caches, many memory blocks can map to the same cache line
  - a cache line may have to be thrown out because some other line needs its place in the cache
  - the same line may have to be brought in immediately after
- Conflict misses can be overcome by using higher associativity
  - 4-way associative instead of 2-way
  - can also try to avoid conflict misses in the program design
- 2:1 cache rule of thumb
  - a direct mapped cache of size *N* has about the same miss rate as a 2-way set-associative cace of size *N/2*

19

Cache trashing

- Repeatedly throws out a cache line that we need in the next memory access
  - can occur with direct mapped and 2-way set associative caches
- Assume we have a direct mapped cache
  - cache line size of 32 bytes (= 8 words)
- Two arrays X and Y contiguosly located cache size apart
  - (address of X) MOD (nr. of lines in cache) = (address of Y) MOD (nr. of lines in cache)
- X[0] and Y[0] are mapped to the same cache line
  - when one is brought in to cache, the other is thrown out
  - causes cache trashing if we access both arrays sequentially



20

Example of cache trashing

- In the first iteration, the reference to X[0] causes a compulsory cache miss
  - the cache line containing X[0] – X[3] is brought in
  - X[0] is placed in a register
- The cache line containing Y[0] – Y[3] is brought in
  - maps to the same line, replaces X[0] – X[3]
  - X[0] is placed in a register

```
SIZE = 64*1024; /* 64K */
double X[SIZE], Y[SIZE];
   . . .
for (i=0; i<SIZE; i++) {
    Y[i] = X[I]+Y[i];
}
```

- X[0] and Y[0] are added and Y[0] is stored
- In the next iteration the cache line containing X[1] has to be brought in again
  - conflict cache misses in all iterations

21

Cache trashing (cont.)

- Can also get cache trashing in 2-way set associative caches
- Three consecutive arrays of cache size
  - X[k], Y[k] and Z[k] all map to the same set
  - the set size is two
  - one will always be thrown out in each iteration
- Can be avoided by padding the arrays
  - insert an array of the cache line size between the arrays
  - X[0], Y[0] and Z[0] map to different cache lines

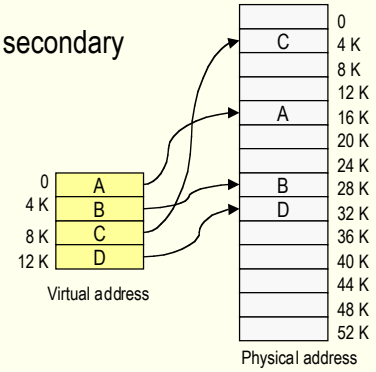
```
SIZE = 64*1024; /* 64K */
double X[SIZE], Y[SIZE], Z[SIZE];
   . . .
for (i=0; i<SIZE; i++) {
    Z[i] = X[I]*Y[i]+Z[i];
}
```

- Trashing may be a problem when array size is a power of two

22

Virtual memory

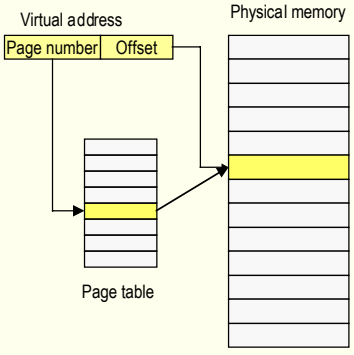
- Decouples addresses used by the program (virtual addresses) from physical addresses
  - the program uses a large contigous address space
  - actual memory blocks may be located anywhere in physical memory
  - some memory blocks may also be on secondary storage
- Memory is divided into pages
  - page size can be from 512 bytes to 4 MB
- Virtual addresses are translated to physical addresses using a page table



23

Page tables

- Stores the mapping of logical to physical addresses
- Indexed by the virtual page number
  - one entry per page in the virtual address space
- Page tables are usually large
  - stored in virtual memory
  - need two virtual-to-physical translations to find a physical address
- Use a translation lookaside buffer (TLB) as a cache for addess translations



24

Translation lookaside buffer

- Cache memory for address translations
  - tag field holds a part of the virtual address
  - data field holds the physical page frame number
  - also status bits: valid, use, dirty
- Implemented by an associative cache memory
- TLB is limited in size
  - virtual addresses not in the TLB cause a TLB miss
- Repeated TLB misses cause very bad performance
  - same as for repeated cache misses
- Good cache behaviour usually implies good TLB behaviour

25