Kiến trúc máy tính nâng cao

Chương 1: Kiến trúc máy tính

1. Chi phí mạch:
2. **CPU**:
   1. Chức năng chính của CPU là thực thi các lệnh đc lưu trong bộ nhớ chính. 1 lệnh bảo CPU thực thi 1..n tác vụ cơ bản của nó.
   2. CPU bao gồm các thanh ghi & bus giao tiếp bên trong để truyền data giữa control unit CU, ALU, registers. CU là bộ giải mã các lệnh để thực thi & điều khiển các thành phần khác làm việc.
   3. CPU bảo gồm 2 thành phần chính:
      1. A Program control unit
      2. An Arithmetic & Logic Unit (ALU)
3. **Registers**:
   1. CPU cần vùng lưu trữ dữ liệu tạm khi làm việc
   2. Các đơn vị lưu trữ này đc gọi là thanh ghi registers.
   3. Chúng là thành phần có level cao I trong cấu trúc bộ nhớ fân cấp
   4. Số lượng & chức năng of các thanh ghi ≠ giửa các máy tính
   5. Thanh ghi có 2 nhiệm vụ chính:
      1. Thanh ghi người dùng thấy đc: đc dùng bởi máy & mã assembly của lập trình viên. Gồm:
         * Các thanh ghi dùng cho mục đích chung
         * Thanh ghi dữ liệu
         * Thanh ghi địa chỉ
         * Thanh ghi điều kiện
      2. Thanh ghi điểu khiển & trạng thái: dùng bởi CU để điều khiển CPU & hệ điều hành để thực thi chương trình.
4. **Machine Instructions**: CPU chỉ có thể thực thi mã máy dạng binary, gọi là những lệnh máy:
   1. ***Một machine instruction cụ thể có các thông tin sau***:
      1. Cái gì phải thực hiện xong
      2. Thao tác này áp dụng cho ai
      3. Nơi lưu kết quả trả về
      4. Địa chỉ lệnh kế tiếp
   2. ***4 dạng của machine instruction***:
      1. Lệnh số học & luận lý
      2. Lệnh chuyển dữ liệu từ bộ nhớ vào thanh ghi của CPU
      3. Điều khiển chương trình
      4. Giao tiếp với thiếc bị ngoại vi
5. **Instruction Set Design**: Thiết kế tập lệnh là quan trọng của hệ thống máy tính.
   1. Những vấn đề quan trọng nhất là:
      1. Bao nhiêu & loại tác vụ sẽ có; độ phức tạp của tác vụ
      2. Kiểu dữ liệu
      3. Định dạng của lệnh: độ dài, số lượng các địa chỉ, kích thước của các trường khác nhau, …
      4. Số lượng các thanh ghi
      5. Mô hình đánh địa chỉ
   2. Các vấn đề này có quan hệ với nhau
6. **Machine Cycles** (chu kỳ máy)
   1. Thực thi một lệnh được thực hiện trong 1 chu kỳ máy
   2. CPU thực thi 1 lệnh sau lệnh khác, lập đi lập lại
   3. Thời gian chu kỳ máy cho biết hiệu năng (speed)
   4. Các lệnh khác nhau cần thời gian khác nhau để thực thi, thời gian thực thi trung bình đc dùng để tính toán hiệu năng
7. **Đánh giá hiệu năng CPU:**
   1. 2 thước đo là:
      1. **Response time** (execution time) là thời gian từ lúc bắt đầu 🡪 khi 1 task hoàn tất. 1 task gồm truy xuất đĩa, hoạt động I/O, OS overhead, CPU execution time… Quan trọng với ***user***
      2. **Throughput** (**bandwidth** thông lượng) tổng số lượng công việc hoàn tất trong 1 thời gian cụ thể. Quan trọng với hệ thống quản lý dự liệu trung tâm như server.
   2. ***Response time & Throughput bị ảnh hưởng bởi***:
      1. *Thay đổi CPU nhanh hơn*: tăng Response time & Throughput
      2. *Tăng số lượng CPU*: tăng Throughput
8. **Hiệu năng Performance (Speed)**:
   1. Hiệu năng lớn khi thời gian thực thi nhỏ nhất:
   2. Nếu X chạy nhanh gấp n lần Y thì
   3. Giảm thời gian đáp ứng (response time) hầu như luôn luôn cải thiện thông lượng (throughput).
9. **Đo hiệu năng**:
   1. ***Elapsed time*** (***wall clock time***, ***respone time***) là tổng thời gian đáp ứng & các yếu tố khác (processing, I/O, OS overhead, idle time). Dùng để xác định hiệu năng của hệ thống
   2. ***CPU time*** (ko kể thời gian truy xuất ổ cứng, I/O): là thời gian CPU thực thi 1 task. Bao gồm thời gian CPU cho người dùng & cho hệ thống. Dùng xác định hiệu năng của CPU
   3. *Different programs are effected differently by CPU & system performance*
10. **CPU Clocking**:
    1. Bộ xử lý bị quản lý bởi 1 clock có tốc độ ko đổi
    2. ***Clock Period*** (Clock Cycle **CC**) thời gian of 1chu kỳ clock.
    3. ***Clock Frequency*** (Clock Rate **CR**): số chu kỳ mỗi giây

CC = 1/CR

Vd: CC = 250ps = 250 x 10-12s => CR = 1/CC = 4 GHz

1. **CPU execution time** (**CPU time**)

or

Có thể tăng hiệu năng bằng cách giảm

* + 1. số lượng clock cycle cần thiếc cho 1 chương trình
    2. hay độ dài của 1 clock cycle

1. VD: **Tính hiệu năng**: Chạy chương trình trên máy tính A 2GHz clock trong 10s. Clock rate của máy tính B bao nhiêu để nó chạy chương trình này trong 6s; biết B cần 1,2 clock hơn máy tính A để chạy chương trình

*Công thức số chu kỳ*: **ClockCycles = CPUTime x ClockRate**.

*Giải*: Số chu kỳ của A:

ClockCycleA=CPUTimeA x ClockRateA=10s x 2GHz=20x109

ClockRateB=1.2xClockCycleA / CPUTimeB = 4GHz

1. **Clock Cycles per Instruction CPI**: (số chu kỳ clock của 1 lệnh)

và

* 1. Ko phải tất cả các lệnh đều có cùng thời gian thực thi lệnh
  2. CPI là trung bình số chu kỳ clock mỗi lệnh cần để thực thi nó

1. VD: **So sánh hiệu năng**: Máy tính A&B hiện thực cùng 1 ISA. A có CC=250ps, CPI=2.0; B có CC=500ps, CPI=1.2. Ai nhanh hơn?

*Giải*: Thời gian chạy chương trình (I Instruction Count) của A & B:

CPUTimeA= I x CPIA x CCA = I x 2.0 x 250ps = I x 500 ps

CPUTimeB= I x CPIB x CCB = I x 1.2 x 500ps = I x 600 ps

=> A nhanh hơn 600/500=1.2 lần B

1. Ví dụ: **CPI trung bình**: có 3 lớp lệnh A, B, C

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Class | A | B | C | **IC** | **CC** | **Avg.CPI** |
| CPI for class | 1 | 2 | 3 |  |  |  |
| IC in sequence 1 | 2 | 1 | 2 | **2+1+2=5** | **2x1+1x2+2x3=10** | **10/5=2.0** |
| IC in sequence 2 | 4 | 1 | 1 | **4+1+1=6** | **4x1+1x2+1x3=9** | **9/6=1.5** |

IC: Instruction Count

Avg.CPI (CPI trung bình) ko kết luận nhanh chậm

1. **Chi tiết CPI**:
   1. Nếu lớp câu lệnh khác nhau thì số lượng chu kỳ cũng khác nhau.

Với:

* + 1. ICi: số lượng (phần trăm) lệnh của class i thực thi
    2. CPIi: số trung bình clock cycles mỗi lệnh của instruction class
    3. n: số lượng instruction class

1. VD: **CPI của các class**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **OP** | **Freq** | **CPIi** | FreqxCPIi | (a) | (b) | (c) |
| ALU | 50% | 1 | 0.5 | 0.5 | 0.5 | 0.25 |
| Load | 20% | 5 | 1.0 | 0.4 | 1.0 | 1.0 |
| Store | 10% | 3 | 0.3 | 0.3 | 0.3 | 0.3 |
| Branch | 20% | 2 | 0.4 | 0.4 | 0.2 | 0.4 |
| CPI trung bình | | | 2.2 | 1.6 | 2.0 | 1.95 |

* 1. Dữ liệu đc cache nên giảm Load còn CPILoad=2 chu kỳ

CPUnew=1.6 x IC x CC; so với lúc đầu thì nhanh hơn 2.2/1.6 lần

* 1. Giảm 1 cycle cho lệnh rẻ nhánh => CPIBranch=1=>nhanh hơn 2.2/2
  2. 2 lệnh ALU có thể thực thi cùng lúc => nhanh hơn 2.2/1.95 lần

1. **Các yếu tố ảnh hưởng đến InstructionCount, CPI, Clock Cycle**:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Instruction Count | CPI | Clock Cycle |
| Algorithm | **X** | X |  |
| Programming language | **X** | X |  |
| Compiler | **X** | **X** |  |
| ISA | **X** | **X** | **X** |
| Core organization |  | **X** | **X** |
| Technology |  |  | **X** |

1. **Power & Energy**:
   1. ***Dynamic power***: CMOS

Powerdynamic=1/2 x CapacitiveLoad x Voltage2 x FrequencySwitched

* 1. ***Mobile devices***:

Energydynamic=CapacitiveLoad x Voltage2

* 1. Fixed task, giảm clock rate (frequency switched), giảm power nhưng ko giảm energy.

1. **Reducing Power**:
   1. CPU mới có:
      1. 85% capacitive (điện dung) load của CPU cũ
      2. Giảm 15% voltage & 15% frequency

CPU mới dùng ít năng lượng hơn CPU cũ

* 1. ***Bức tường năng lượng*** (***power wall***): chúng ta ko thể giảm voltage hay nhiệt thêm nữa
  2. ***Static Power***: vì rò rỉ xuất hiện ngay cả khi transitor tắt:
     1. Rò rỉ tăng lên trong bộ xử lý khi kích thước transistor nhỏ hơn
     2. Tăng số lượng transistor=> tăng tiêu thụ năng lượng ngay cả khi chúng bị tắt.
     3. ***ExecutionTimeRation*** = ***SPEC\_ration*** = ***RefTime/ExTime***
     4. Geometric mean:

ExecutiveTime=CPUTime=IC x CPI x CC

1. **SPEC Power Benchmark**: mức độ tiêu thụ năng lượng khác nhau ở các mức workload khác nhau.
   * 1. Performance: ssj\_ops/sec
     2. Power: Watts (Joules/sec)

=>

1. **Time maket**:
   * 1. Product life = 2W, đỉnh ở W
     2. Ontime = ½ \* 2W \* W
     3. Delayed = ½ \* (W-D+W)\*(W-D)
     4. Phần trăm lợi nhuận bị mất là (D(3W-D)/)
2. **Computer design must meet**:
   * 1. Functional requirements
     2. Area/performance/cost/power goals: tối ưu, ước lượng, tìm kiến trúc khả thi nhất
     3. Cân nhắc các nhân tố khác: thời gian ra sản phẩm, định hướng công nghệ, …
3. **Speeding it up**:
   * 1. Pipelining
     2. On board cache; on board L1 & L2 cache
     3. Dự đán lệnh rẽ nhánh
     4. Phân tích dòng dữ liệu
     5. Thực hiện theo phỏng đoán
4. **Performance Balance**:
   * 1. Tăng tốc bộ xử lý
     2. Tăng dung lượng bộ nhớ
     3. Memory speed lags behind processor speed
5. **Thu hẹp khoảng cách tốc độ bộ nhớ & CPU**:
   * 1. Tăng số lượng bit truy xuất trong 1 lần đọc/ghi. Tạo DRAM “wider” (rộng) hơn là “deeper” (sâu)
     2. Change DRAM interface
     3. Reduce frequency of memory access. More complex cache & cache on chip
     4. Increase interconnection bandwidth:
        + High speed buses
        + Hierarchy of buses

Chương 2: **Hệ thống bộ nhớ**

1. **Memory characteristics**:
   1. ***Location***:
      1. Internal: lưu các dữ liệu đang được CPU dùng
      2. External: vùng nhớ lâu dài & lớn; lưu trử data & progam. Truy xuất qua I/O controllsers
   2. ***Capacity*** (sức chứa)
      1. *Word size*: số lượng bit dùng biểu diễn 1 byte; =độ dài câu lệnh
      2. *Number of words* (or bytes)
   3. ***Unit of transfer***:
      1. Các phần tử có thể đc định địa chỉ
      2. Đơn vị giao tiếp: số lượng bit có thể đọc/ghi bộ nhớ trong 1 lần; ko nên bằng 1 word hay 1 addressable unit.
      3. Internal: thường bị chi phối bởi độ rộng của bus
      4. External: thường 1 block lớn hơn 1 word.
   4. ***Access method***:
      1. *Sequential* (tape): bộ nhớ đc tổ chức thành các đơn vị data (records), bắt đầu ở điểm đầu & đọc tuần tự. Thời gian truy xuất dựa vào vị trí của data & vị trí trước đó.
      2. *Derect* (disk): các block độc lập có địa chỉ duy nhất; truy cập bằng cách nhảy tuần tự tìm kiếm. Thời gian truy xuất phụ thuộc vào vị trí data & vị trí trước đó.
      3. *Random* (Ram) Individual addresses identify locations exactly; Access time is independent of location or previous access.
      4. *Associative* (Cache) Data is located by a comparison with contents of a portion of the store. Access time is independent of location or previous access.
   5. ***Performance***:
      1. *Thời gian truy xuất* (Access time):
         * Của RAM là từ khi có địa chỉ đến khi data sẳn sang
         * For non-RAM, it’s the time between positioning the read/write mechanism at the desired location
      2. *Memory cycle time*:
         * Thời gian cần cho việc “phục hồi” trước lần truy xuất kế
         * **Cycle time = Access + Recovery**
      3. *Transfer Rate*: tốc độ mà data vào ra 1 memory unit
         * For RAM: **transdfer rate = 1/(cycle time)**
         * For non-RAM: **TN = TA + N/R**
           1. TN: thời gian trung bình để đọc ghi N bits
           2. TA: thời gian trung bình để truy xuất
           3. N: số lượng bit
           4. R: tốc độ truyền, đơn vị bps
   6. ***Physical type***: Semiconductor (RAM0, Magnetic (Disk & Tape), Optical (CD, DVD), Others (Buble, Hologram)
   7. ***Physical characteristics***:
      1. Decay:
      2. Volatility: tính bay hơi
      3. Erasable: xóa đc
      4. Power consumption: tiêu thụ năng lượng
   8. ***Organization***: Cách sắp xếp những bít vào word, ko phải lúc nào nó cũng rỏ rang. Giống như xếp sách
2. **Thuộc tính quan trọng của bộ nhớ**:
   1. Speed: tốc độ càng nhanh càng tốt
   2. Size: kích thước càng lớn càng tốt
   3. Cost: Chi phí hợp lý

=> Chúng ta phải xác định công nghệ dùng để hiện thực bộ nhớ

1. **Main memory**:
   1. MM có thể lưu như 1 tập hợp các tế bào nhớ có thể nhớ 1 word.
   2. Mỗi cell đc gán 1 địa chỉ duy nhất & địa chỉ này đc đánh số 0…n
   3. Số lượng bit đại chỉ xác định kích thước của bộ nhớ (16 bits-65K, 24 bits-16M; 32 bits-4G)
   4. ***Memory control unit*** has a read/write control mechanism & an address selection mechanism.
2. **Memory Bandwidth**: biểu diển số lượng data có thể truy xuất từ bộ nhớ trong 1 giây.

M=Bandwidth = AmountOfDataPerAccess / MemoryCycleTime

* 1. *Vd*: MCT=100ns & 4 bytes (a word) per access

=>M-bandwidth = 40MBps

* 1. *Có 2 kỹ thuật cơ bản để tăng bandwidth của 1 bộ nhớ*:
     1. Giảm memory cycle time: nhưng đắc & size bộ nhớ bị giới hạn
     2. Chia bộ nhớ thành nhiều bank,mỗi bank chứa control unit riêng

1. **Memory Hierarchy**:
   1. *Phân cấp*:
      1. Registers: trong CPU
      2. Internal or Main memory: bao gồm nhiều cấp cache; RAM
      3. External memory: Backing store
   2. Khi duyệt bộ nhớ phân cấp từ trong CPU ra ngoài thì:
      1. Giảm chi phí/bit (cost/bit)
      2. Tăng dung lượng
      3. Tăng thời gian truy xuất
      4. Giảm mật độ truy xuất bởi CPU.
2. **Thời gian truy xuất bộ nhớ trung bình**:

AverageAccessTime ≈ Phit x Tcache\_access + (1-Phit) x (Tmm\_access+ Tcache\_access) x BlockSize + Tchecking

Với

* + 1. Phit = khả năng cache hit; tỉ lệ cache hit
    2. Tcache\_access= cache access time
    3. Tmm\_access= main memory access time
    4. BlockSize = số lượng word trong cache block
    5. Tchecking = thời gian cần thiếc cho việc kiểm tra cache hit or miss

VD: A computer has 8MB MM với 100ns access time, 8KB cache with 10ns access time, BS=4; Tchecking=0; Phit=0.97 => **AAT=22.9 ns**

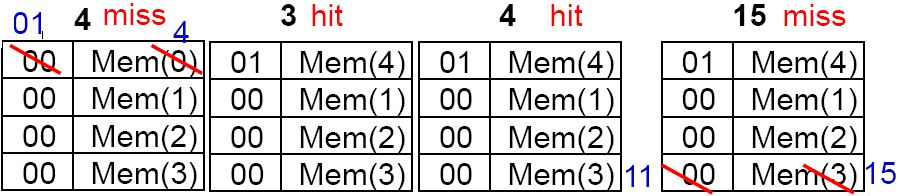
1. **Thuật ngữ trong Memory Hierachy**:
   1. **Hit**: data ở block trong upper level memory
      1. *HitRate*: là tốc độ truy xuất upper để phát hiện data ở đó
      2. *HitTime*: thời gian truy xuất upper level memory; bao gồm RAM access time + Time to determine hit/miss
   2. **Miss**: data ko ở trong upper mà ở trong lower level memory:
      1. *MissRate* = 1 – HitRate
      2. *MissPenalty*: Time thay thế 1 block trong upper + Time chuyển block vào Processor
      3. HitTime ≪ MissPenalty
2. **Cache Design**: kích thước & số block đc copy phải đc thiết kế cẩn thận, cũng như giải thuật quyết định bỏ block nào ra khỏi cache khi cache đầy.
   * 1. Cache block size
     2. Total cache size
     3. Mapping function
     4. Replacement method
     5. Write policy
     6. Number of caches: single, 2 or 3 level; Unified >< Split cache
3. **Mapping Function**:
   1. Cần để ánh xạ bộ nhớ chính vào 1 dòng cache & xác định khối bộ nhớ chính nào đang đc gắn với 1 cache line
   2. Chọn hàm ánh xạ là xác định cách tổ chức cache
   3. Ex:
      1. Cache 64Kbytes; CacheBlock=4bytes =>có 16k (214) block line
      2. Data are transferred in blocks of 4byte. 16MByte main memory

=> 4M block, mỗi block 4 byte

1. **Direct Mapping Cache**: mỗi block of bộ nhớ chính đc ánh xạ vào 1 cache slot cố định. Vì vậy một số phần tử ở lower level phải chia sẽ vị trí với upper level.
   1. ***Công thức***: i = j modulo m
      1. i = cache line number
      2. j = main memory block number
      3. m = number of lines in the cache.
   2. ***Cache line***: đc đánh thứ tự ***Index***. Chia thành 3 phần:
      1. ***Valid***: xác nhận data trong cache line có hợp lệ hay ko
      2. ***Tag***: đc tính bằng công thức ***tag = i = j modulo m***
         * i = Số thứ tự của cache line
         * j = số lượng block trong bộ nhớ chính
         * m = số lượng line trong cache
      3. ***Data***
   3. ***Địa chỉ***: độ dài là **S+W**, đc chia làm 2 phần:
      1. W bit xác định duy nhất 1 word (hay byte offset) trong 1 block of bộ nhớ chính. Word (hay byte) này trong block sẽ đc dùng.
      2. S bit xác định 1 khối bộ nhớ (bộ nhớ chính có 2S block)
         * A cache line field: R bit; chứa giá trị ***index*** của cache line
         * A tag: S-R bits; chứa giá trị modulo với m.
      3. VD: 24 bits địa chỉ, trong đó có 2 bit xác định word (4 byte block), 22 bit xác định block (8 bit tag & 14 bit slot or line)

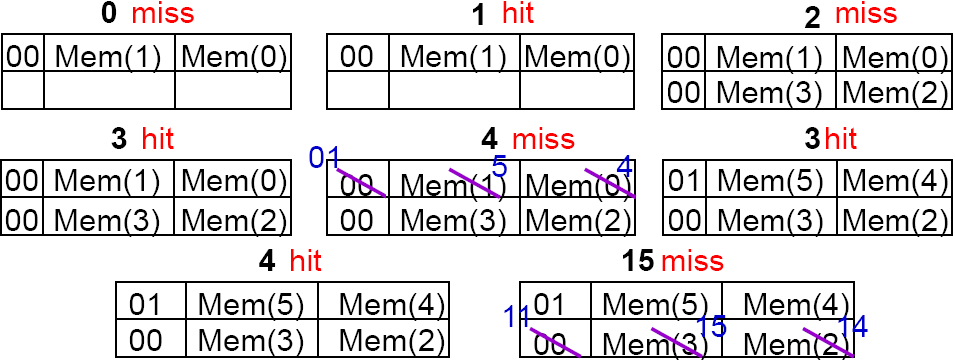
|  |  |  |
| --- | --- | --- |
| Tag (**S-R** bits): 8 | Line/Slot/Index (**R** bits): 14 | Word (**W** bits): 2 |

* + - * Ko có 2 block trong cùng 1 dòng có cùng Tag field.
      * Kiểm tra nội dung cache bằng cách tìm line & kiểm tra Tag.
  1. ***So trùng***: CPU phân tích Address:
     1. Từ giá trị ***Index*** của Address, CPU lấy cache line tương ứng
     2. Nếu cache line đó ***Valid*** & ***TagAddress=TagCacheLine*** thì Cache line này đc chọn;
     3. Vì BlockSize = LineSize nên trả về ***Data*** = CacheLine.
  2. ***Pros & Cons***:
     1. Hiện thực đơn giản nên ko đắc tiền.
     2. Luân phiên truy xuất 2 block đc ánh xạ vào cùng 1 cache slot sẽ làm cache miss rate tăng cao => gọi là hiện tượng ***thrashing***.
  3. ***Summary***:
     1. Độ dài địa chỉ: (S+W) bits
     2. Số lượng phần tử có thể đánh địa chỉ = 2S+W words hay bytes
     3. BlockSize = LineSize = 2W words or bytes
     4. Số lượng Block trong bộ nhớ chính = 2S+W/2W = 2S
     5. Số lượng Line trong cache = m = 2R
     6. Kích thước của tag = (S-R) bits



4 = **01**00 15 = **11**11

1. **Multiword Block Direct Mapped Cache**:
   1. Giống Direct Mapped Chache nhưng Data của cache line gồm nhiều Word. Khi đó trường Word (Byte offset) của địa chỉ chứa index của Word cần đọc ra.
   2. Do đó, khi ***Valid*** & ***Tag*** hợp lệ, W bits của trường ***Word*** trong Address sẽ xác định Word cần truy xuất; trả về cho Data



* + 1. Đọc 1 lần 2 ô nhớ Mem(i+1), Mem(i) vào cache block.
    2. Tag = i div 2W; W là số bit của phần Word. Vd: 15 = **11**11

1. **Associative Mapping**:
   1. 1 block bộ nhớ chính có thể đc load vào bất kỳ slot nào của cache
   2. Memory Address = Tag + Word
   3. Tag xác định Block of Memory duy nhất
   4. Để xác định nếu 1 block trong cache ko, 1 thiếc bị cần để kiểm tra mỗi slot’s tag => mất thời gian cho quá trình tìm kiếm
   5. ***Ex: Address structure & Cache Line Structure***:
      1. *Address*: 22 bit tag lưu trữ block data dài 32bits=4byte=2word

|  |  |
| --- | --- |
| Tag: 22 bit | Word: 2 bit |

* + 1. *Cache line*: 16Kline Cache

|  |  |
| --- | --- |
| Tag: 22 bit | Data: 32 bit |

* + 1. So sánh TagAddress & TagCacheLine để kiểm tra *hit*.
    2. 2 bit of address xác định 16 bit word trong 32 bit data block
    3. Ex:
       - Data 32 bits có giá trị là FEDCBA98
       - Address 16339C
       - Tag = Address div 2W = 058CE7
  1. ***Summary***:
     1. Address length = (S+W) bits
     2. Số lượng phần tử có thể đánh địa chỉ: 2S+W words/bytes
     3. BlockSize = LineSize = 2W words/bytes
     4. Số lượng blocks trong bộ nhớ chính = 2S+W/2W=2S
     5. Số lượng line trong cache ko xác định đc
     6. Kích thước Tag = S bits

1. **Set Associative Mapping**: Miss Rate tỉ lệ nghịch số tập hợp v
   1. Cache đc chia thành v tập hợp; mỗi tập hợp chứa k slots:

m = k \* v

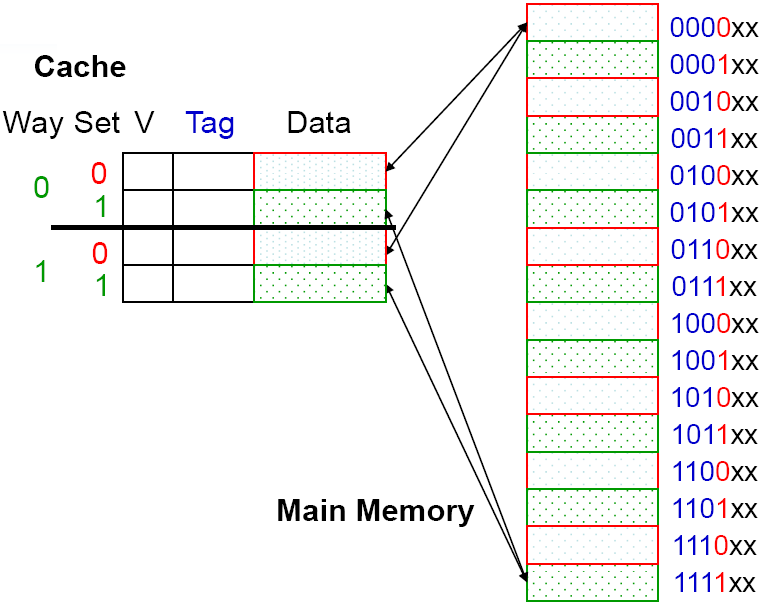
* 1. Công thức xác định Cache Set: **i = j modulo v**. Với block j có thể là bất kỳ trong Cache Set i.

*Ex*: 2 slot mỗi set (k=2); block có thể là 1 trong 2 block của set

* 1. *Direct mapping*: ***k=1***; *Full associative*: ***v=1*** (k = số slot of cache)
  2. *Ex*: 13 bit set number =>Số block trong bộ nhớ chính modulo 213.

Các block 000000, 008000, 00A000, 00C000,…,FF8000 ánh xạ vào cùng 1 Cache Set 0.

* 1. ***Ex: Address structure & Cache Line Structure***: v =2
     1. Address: Tag + Set + Word
        + *Tag*: (s-d) bit; các bit cao xác định TagAddress
        + *Set*: d = lg(v) = 1 bit xác định cache set. SetNumber=AddressNumber div
        + *Word*: w = 32 bit thấp nhất định nghĩa word blocks
     2. So trùng tất cả TagCache trong 1 Cache Set với TagAddress để xác định memory block có trong cache hay ko (hit/miss)



* 1. *Ex: Tag 9 bit; Set 13 bit; Word 2 bit*
     1. Address = 167FFC;
     2. Tag = 02C (9 bit đầu)
     3. SetNumber = 1FFF (13 bit kế)
  2. ***Summary***:
     1. Độ dài địa chỉ = (s+w) bits
     2. Số lượng đơn vị có thể đánh địa chỉ: 2S+W words/bytes
     3. BlockSize=LineSize=2W words/bytes
     4. Số lượng blocks trong bộ nhớ chính: 2S+W/2W=2S
     5. Số lượng line trong 1 tập hợp = k
     6. Số lượng tập hợp = v = 2d
     7. Số lượng line trong cache = k.v = k. 2d
     8. Kích thước của tag = (s-d) bits

1. **Replacement Algorithms** (RA)
   1. Với direct mapping thì ko cần RA
   2. Với associative mapping, RA cần để xác định block cần replace:
      1. Dùng phần cứng để hiện thực giải thuật => tăng tốc độ
      2. Least-recently used (***LRU***) thay block ở trong cache mà lâu nhất ko đc dùng đến
      3. First in first out (***FIFO***)
      4. Least-frequently used (***LFU***) thay block ít đc tham khảo nhất
      5. Random
2. **Write Policy**:
   1. ***Vấn đề***:
      1. Duy trì sự nhất quán giửa cache content & main memory content với chi phí performance thấp
      2. Ko overwrite cache block nếu memory of nó vẫn chưa cập nhật
      3. I/O có thể định địa chỉ trực tiếp bộ nhớ chính
      4. Nhiều CPU phải có các caches độc lập nhau
   2. ***Write hit***:
      1. *Cho phép cache & memory ko giống nhau* (***write-back***)
         * Ghi data của cache block này chỉ khi nó bị thay thế bởi block khác (ghi vào next level trong memory hierarchy)
         * Cần ***dirty*** bit cho mỗi cache block để biết nếu nó cần ghi ra memory khi bị “đuổi ra” khỏi cache ko
      2. *Bắt buộc cache memory giống nhau*:
         * Luôn luôn ghi data cả cache block & next level in the memory hierarchy (***write-through***) => ko cần ***dirty*** bit
         * Ghi vào next level sẽ chậm => dùng ***write buffer***, chỉ khi buffer đầy thì mới ghi
   3. ***Write Buffer for Write-Through Caching***:
      1. *Mô hình*:

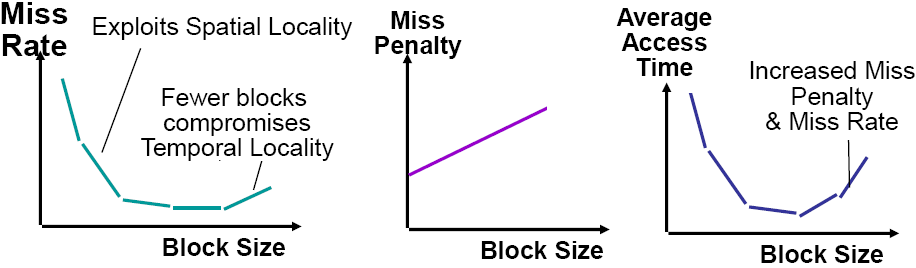
|  |  |
| --- | --- |
|  | * + - * Processor: ghi data vào cache & main memory       * Memory Controller: ghi dữ liệu của buffer vào memory |

* + 1. Write buffer chỉ là 1 FIFO; hoạt động tốt nếu

StoreFrequency (w.r.t time) ≪ 1/DRAM write cycle

* + 1. Vấn đề write buffer là StoreFrequency 🡪1/DRAM write cycle làm write buffer trở nên bảo hòa (saturation). Giải pháp là dùng 1 write-back cache; hay dùng L2 cache

1. **Cources of Cache Misses**:
   1. ***Compulsory***
   2. ***Confilict***: nhiều ô nhớ ánh xạ vào cùng 1 vị trí cache. Giải pháp: Tăng cache size hay tăng associativity.
   3. ***Capacity***: cache ko thể có tất cả các block của 1 chương trình => tăng cache size
2. **Cache Coherence**:
   1. **Software Solution**:
      1. Dựa vào phân tích code: xác định code ko an toàn khi caching, đánh dấu chúng để biết ko cached chúng => thêm code để làm điều này
      2. Compiler & OS thỏa thuận để giải quyết điều này
      3. Tăng Compile Time; fức tạp khi truyền từ hardware →software
      4. Giải pháp này thích hợp với các quyết định thận trọng
   2. **Hardware Solution**:
      1. Cache coherence protocols
      2. Dynamic recognition of potential problems
      3. Run time
      4. More efficient use of cache
      5. Transparent to programmer
      6. Two main categories:
         * ***Directory protocols***:
           1. lưu trữ & duy trì thông tin về copy của data trong cache.
           2. Directory lưu trong main memory.
           3. Yêu cầu kiểm tra directory
           4. Dy chuyển dữ liệu phù hợp sẽ đc thực thi: giửa memory & cache hay giữa các cache
           5. Tạo thắc cổ chai ở trung tâm
           6. Hiệu quả với scale system lớn với lược đồ kết nối bên trong phức tạp
         * ***Snoopy protocols***: phân phát cache coherence cho các cache controllers. Cache nhận ra 1 line đc shared; update sẽ cảnh báo các caches khác. Tăng bus traffic. Phù hợp với bus của multiprocessor
3. **Miss Rate, Block Size & Cache Size**:



AverageMemoryAccessTime = HitTime + MissPenalty x MissRate

1. **Do hiệu năng của Cache**:
   1. 2 kỹ thuật để cải thiện hiệu năng cache:
      1. Giảm missrate bằng cách giảm khả năng 2 memory block khác nhau cùng chiếm 1 vùng
      2. Giàm miss penalty bằng cách tăng thêm level của hierarchy
   2. Hiệu năng:

CPUTime = IC x CPI x CC = IC x CPIstall x CC

CPIstall = CPIideal + MemoryStallCycles

MemoryStallCycles = MissRate x MissPenalty

ReadStallCycles = Reads/Program x ReadMissRate x ReadMissPenalty

WriteStallCycles=(Writes/Program x WriteMissRate x WriteMissPenalty) + WriteBufferStalls

* 1. *Vd*: CPIideal = 1.1; 50% arith/logic, 30% ld/st, 20% control. 10% data memory miss with 50 cycle miss penalty

CPI = CPIideal+AverageStallsPerInstruction

=1.1(cycle) + (0.30(datamemops/instr) x 0.10(mis/datamemop) x 50(cycle/miss) )

= 1.1 cycle + 1.5 cycle = 2.6 cycle

* + 1. Vậy 1.5/2.6=58% thời gian CPU bị stalled chờ truy xuất bộ nhớ
    2. 1% miss rate làm CPI tăng 1.5/10 = 0.5.
  1. *Vd*: CPIideal=2; 100 cycle miss penalty; 36% load/store instr’s, 2% I$; 4% D$ miss rate.

MemoryStallCycles = 2% x 100 + 36% x 4% x 100 = 3.44

CPIstall = 2 + 3.44 = 5.44

1. **Reducing Cache Miss Rates**:
   1. *Cho phép các block ánh xạ linh động hơn*: fully associative cache hay n-way set associative.
   2. *Dùng nhiều mức cache*: CPIideal=2; 100 cycle miss penalty (to main memory); 36% load/store, 2% L1$ & 4% D$ miss rate, UL2$ (cho lệnh & data) có 25 cycle miss benalty & 0.5% miss rate.

CPIStalls= 2 + .02x25 + 0.36x.04x25 + 0.05x100 + .36x0.05x100 = 3.54

CPIStalls\_NoL2= 2 + 0.02x100 + .36x.04x100 = 5.44

1. **Vấn đề cân nhắc khi thiết kế Cache nhiều mức**:
   1. Thiết kế caches L1 & L2 khác nhau:
      1. Cache đầu tiên nên tập trung giảm đến mức ít nhất ***hit time*** để hổ trợ shorter clock cycle. Smaller with smaller block sizes
      2. Cache thứ 2 nên tập trung giảm ***miss rate*** để giảm penalty thời gian truy xuất bộ nhớ chính. Larger with larger block sizes.
   2. Miss penalty của L1 cache đc giảm xuống bởi có mặt của L2 cache. Do đó, L1 có thể smaller & faster nhưng miss rate cao
   3. L2 cache thì hit time ít quan trọng hơn miss rate:
      1. L2$ hit time xác định bởi L1$ miss penalty
      2. L2$ local miss rate ≫ global miss rate
2. **Improving Cache Performance**: Slide 125-126
   1. *Giảm hit time của cache*: Cache nhỏ hơn; Cache ánh xạ trực tiếp; Blocks nhỏ hơn; for writes
   2. *Giả miss rate*: cache lớn hơn; chiến lược ánh xạ linh động; các block lớn hơn; Victim cache (small buffer holding most recently discarded blocks)
   3. *Reduce the miss penalty*: smaller blocks, dùng write buffer …
3. **RAM**: truy xuất ngẩu nhiên, Read/.Write; bay hơi; lưu trử tạm thời; tĩnh hay động.
4. **Transistor**:
   1. nMos: tích cực dương (dẩn khi gate = +)
   2. pMos: tích cực âm (dẩn khi gate = -)
5. **Lưu trữ byte Syndrome word**:
   1. Mỗi bit của syndrome là 0|1; nếu bit i =1 thì bit datai lổi
   2. K bit syndrome word: giá trị từ 0..2K-1
      1. 0 nếu ko có lổi đc phát hiện
      2. 2K-1 giá trị xác định bit bị lổi
   3. Lổi có thể bị phát hiện trong M data bits hay K check bits
      1. 2K - 1 ≥ M+K
      2. Vd: 1 word 8 data bits (M=8), K ≥ 4
6. **Characteristics of Syndrome**:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Position | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|  | 1100 | 1011 | 1010 | 1001 | 1000 | 0111 | 0110 | 0101 | 0100 | 0011 | 0010 | 0001 |
| Data bit | D8 | D7 | D6 | D5 |  | D4 | D3 | D2 |  | D1 |  |  |
| Check bit |  |  |  |  | C8 |  |  |  | C4 |  | C2 | C1 |

1. **A**:
2. **A**:
3. **A**:

a