Contents

1 Final Review

1.1 RAM

Is "random" because you can access any byte in any order

- Baisc unit of storage is a cell
  - 1 bit = 1 cell
  - multiple chips form memory

1.2 SRAM (Static RAM)

- 6 transitors per bit
- Very stable, no leakage
- Very fast
- More expensive

1.3 DRAM (Dynamic RAM)

One transistor per bit

- can have much larger memory per cost
- Charge stored on a capicitor, which will leak after a while
  - Therefore the controller has to habitually read entier memory and refresh memory

1.4 Nonvolatile Memory

- SRAM and DRAM are volatile memory
  - Will lose data if powered off

1.4.1 Read-only memory (ROM)

Code is hardcoded into the circuitry of the chip.

- Thus, it is nonvolatile and will never change

1.4.2 Flash memory

- Electrically erasbale programmable ROM, with partial erase capability.
  - Wears out after 100,000 erasings

1.5 Memory Read Transaction

1.6 Harddisk Geometry

- Disk consists of platters eachwith two surfaces
- Each track consists of multiple tracks (concentric circles)
- Each track is divided into sectors, which are the basic unit of the disk

1.7 Disk access time

- Average time to access some target sector:
  - $T_{access} = T_{avgseek} + T_{avgrotation} + T_{avgtransfer}$
- Seek time for head is biggest limiting factor
- At 7200 RPM, $T_{access} = 13.02$ms
- SRAM takes about 4ns to access a 4-bit word

1.8 SSD

- Organized into pages and blocks
  - Page: 512-4kbs
  - Blocks: 32-128 pages
  - data read/written in units of pages
- Reads much faster tahn HDD
- Erasing takes long though

1.9 Cache

1.9.1 Cache size

C = S x E x B

- C - cache
- S - number of sets
- E - number of lines per set
- B - block size

1.10 Virtual address

1.10.1 Basic parameters

- N: 2^n number of addresses in V
- M = 2^m number of addresses in P
- P = 2^p page size

1.10.2 Components of a VA

- TLBT (TLB tag)
  - which page in in the set?
- TLBI (TLB index)
  - which set in the TLB?
- VPO: virual page offset (p bits)
  - index into the page itself (which byte of memory do we want?)
- VPN: Virtual page number (n-p bits)
  - index into page table

1.11 Control Hazards

1.11.1 Missed Prediction

When a coditional jump is executed is executed prematurally

- Fixed by replacing instructions in decode and execute stages with bubbles.
- instructions effectively cancelled.
- Waste two cycles

1.11.2 Return

Return address is saved on the stack, so need to wait till MEMORY stage is complete

- solved by simply placing a few bubbles until address is retrieved.
- Waste 3 cycles

1.11.3 Load/use

When we retrieve something from memory and the destination of this call is being used as the source in the next call

1.12 Control conditions

Can have combnations of hazards

- Combination A
  - jump in the EXECUTE stage and a return in the Decode
  - Combination B
    - loading is in EXECUTE stage and a return in the DECODE stage (use)

1.13 Exceptions

1.13.1 Causes

- Halt instruction
- Bad address for instruction for data
- Invalid Instruction

1.13.2 Detection

Sometimes an exception will be detected but it shouldnt be triggered

1.13.3 Maintaning exception ordering

- Add status field to pipeline register

Set by FETCH:

- AOK: ok
- ADR: when bad fetch address Also set by MEMORY
- HLT: Halt instruction
- INS: Illegal instruction

1.13.4 Avoiding sideeffects

- Invalid inst. are converetd into bubbles

1.14 Pipelined Implemention *<2015-11-03 Tue>*

1.14.1 Basic Idea

- Divide process into independent stages
- multiple objects being processed at different stages

1.14.2 Throughput

- Number of objects processes per unit time

1.14.3 Latency

Time it takes to process a single object

1.14.4 3-way pipelines version

Break up computation into 3 stages. Save each stage at a register.

- This allows the first stage, for example, to do more computations more quickly, because it has been freed up by the register
- Results in much larger throughput, but higher latency

1.14.5 Limitations

1. Nonuniform Delays
    - Throughput limited by slowest stage
    - Other stages then sit idle for a lot of time
    - Difficult to partion system into balanced system
2. Register overhead As you try to deepen the pipeline (add more stages), the writing to registers becomes more significant
    - Therefore pipelining is limited by the write speed of the registers.

1.14.6 Data Dependency

If operations in stage A depend on results of stage C, stage A depends on C

- Therefore pipeline needs to handle this properly, or risk getting the computaton wrong

1.14.7 Pipline architure stuff

1. S_{field} Value of field held in stage S pipeline registers
2. s_{field} Value of field computed in stage S

1.14.8 Feedback path

Anywhere that a particular value goes back to a particular stage

1.14.9 PC prediciton strategy

1. Instructions that dont transfer control
    - Predict next PC to be valP
    - always reliable
2. Call and Unconditional jumps
    - Predict next Pc to be valC

1.14.10 Stalling for data dependencies

1. Bubble If instruction follows too closely after one that writes to a register, slow it down.
    - hold instruction in decode
    - dynamically inject nop into execute stage
    - stalled instruction is held in DECODE stage
    - Sucessive instructions held in the FETCH stage

1.14.11 Implementing Stalling

1.14.12 Data forwarding

ValE or ValM have been calculated in EXECUTE stage, but have yet to be written to a register

- Therefore, therefore the write stage can be bypassed, by forwarding these values to earlier stages if they need them
- eliminates the need fro nops and bubbles

1. Priority If for some reason there are multiple choices, the earlies stage has priority (EXECUTE has highest priority)
2. Limitation When a register that gets its value from a memory location, and a following instruction needs the value in the registers
    - have to wait to the memory stage to get the value
    - need a bubble
    -

1.14.13 Control Hazards

1. Missed Prediction When a coditional jump is executed is executed prematurally
    - Fixed by replacing instructions in decode and execute stages with bubbles.
    - instructions effectively cancelled.
    - Waste two cycles

2. Return Return address is saved on the stack, so need to wait till MEMORY stage is complete
    - solved by simply placing a few bubbles until address is retrieved.
    - Waste 3 cycles
3. Load/use When we retrieve something from memory and the destination of this call is being used as the source in the next call

1.14.14 Control conditions

Can have combnations of hazards

- Combination A
    - jump in the EXECUTE stage and a return in the Decode
    - Combination B
        * loading is in EXECUTE stage and a return in the DECODE stage (use)

1.14.15 Exceptions

1. Causes
    - Halt instruction
    - Bad address for instruction for data
    - Invalid Instruction
2. Detection Sometimes an exception will be detected but it shouldnt be triggered
3. Maintaning exception ordering
    - Add status field to pipeline register

    Set by FETCH:

    - AOK: ok
    - ADR: when bad fetch address Also set by MEMORY
    - HLT: Halt instruction
    - INS: Illegal instruction
4. Avoiding sideeffects
    - Invalid inst. are converetd into bubbles

1.14.16 Performance Metrics

1. Clockrate
    - Measured in GhZ
2. Rate at which instructions executed
    - CPI: cycles per instruction
        - on avergae, how many clock cycles does each instruction recquire?
    - in PIPE architecure we want as close to 1 instuct. per clock cycle.