Data Manipulation Instructions ("add" Shown Here) (Review)

- Cycle 1

  PC → bus3, load MAR
  clear MD, set MS, set Rd, set Sz

- Cycle 2…n-1

  while (MD == 0) do nothing

- Cycle n

  MDR → bus3, load IR
  inc → PC, load PC
  clear MS

fetch instruction

---

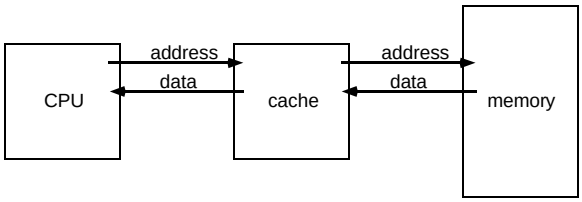
decode and execute

- Cycle n+1

  Reg[src1] → bus1
  Reg[src2] → bus2
  select ALU add operation
  ALU → bus3
  load Reg[dest]

Speeding up the Fetch / Execute Cycle Using Instruction Caching

- A ***small***, ***fast***, memory called the *cache* sits between the CPU and memory



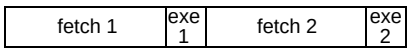
- When the CPU needs data or an instruction, it asks the memory
- The cache *intercepts* that request, and looks in the cache first for the data
  - If it's there (this is called a *hit*), the data is fetched from the cache
  - If it's not there (this is called a *miss*), the cache sends the request to the memory, and the data is fetched from the memory
    - On the way back to the CPU, it's also stored in the cache as well

Speeding up the Fetch / Execute Cycle Using Instruction Caching

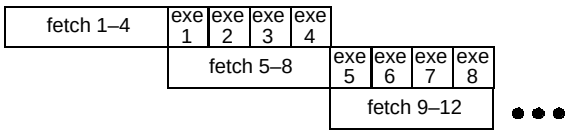
- Why does caching work?
  - Principle of Locality of Reference says:
    - (Temporal locality) Once a piece of data has been fetched, it's likely that *it will be needed again* in the near future
- Further improvements:
  - Principle of Locality of Reference says:
    - (Spatial locality) Once a piece of data has been fetched, it's likely that *others near it will be needed* in the near future
  - Instead of just fetching that one piece of data, fetch a whole block of data
  - More on this in *Computer Architecture*, along with details on how a cache actually works

Speeding up the Fetch / Execute Cycle Using Better Memories

- Normal execution



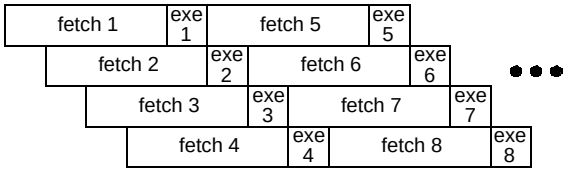
- Wide memories



  - If memory cycle is 4x processor cycle, make memory 4x wider
    - Each memory fetch (taking same amount of time as before) returns 4 instructions
    - Process 4 instructions while fetching the next 4 instructions
  - Disadvantages:
    - Writing to a single word is awkward
    - Branches / jumps can cause problems

Speeding up the Fetch / Execute Cycle Using Better Memories (cont.)

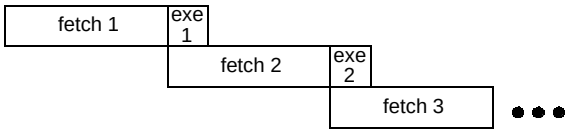
- Interleaved memories (see also p. 82)



  - If memory cycle is 4x processor cycle, use 4 *banks* of memory, with consecutive words in different memory banks (hence the term "interleaved" memories)
    - Overlap memory accesses to each bank
  - Disadvantages:
    - Branches / jumps can cause problems

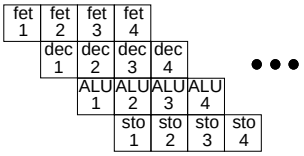
Speeding up the Fetch / Execute Cycle Using Prefetching & Overlapping

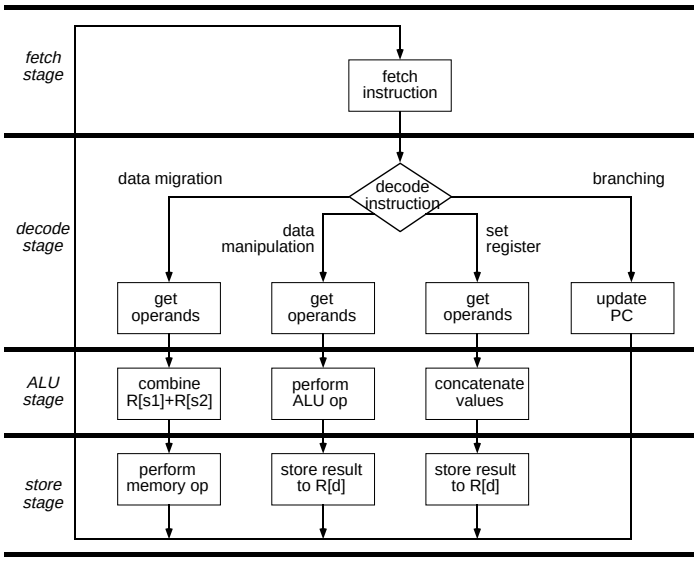
- Instruction prefetching
  - Put a FIFO queue (called an instruction buffer) between instruction fetch logic and instruction execute logic
    - Whenever memory is idle, fetch the next instruction and store it in instruction buffer
  - Disadvantages:
    - Branches / jumps can cause problems
- Overlapped execution



  - Overlap execution of instruction *i* with execution of instruction *i* + 1

Pipelined Instruction Decode / Execute Loop





Pipelining

- Goal: finish executing one instruction every clock cycle
- Divide the fetch / execute loop into several (in this case, 4) *pipeline stages*
  - Each instruction passes through all 4 stages in sequence
  - Each instruction requires 4 clock cycles to execute
- Once the pipeline gets filled, in each clock cycle:
  - A new instruction is fetched
  - An instruction is decoded, and an ALU operation performed
  - An instruction's result is stored, and the instruction finishes its execution