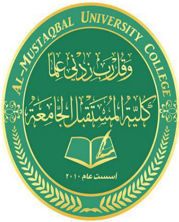


Department of Computer Engineering Techniques (Stage: 4)

Advance Computer Technologies

Dr.: Hussein Ali Ameen

hussein_awadh@mustaqbal-college.edu.iq

Updating main memory

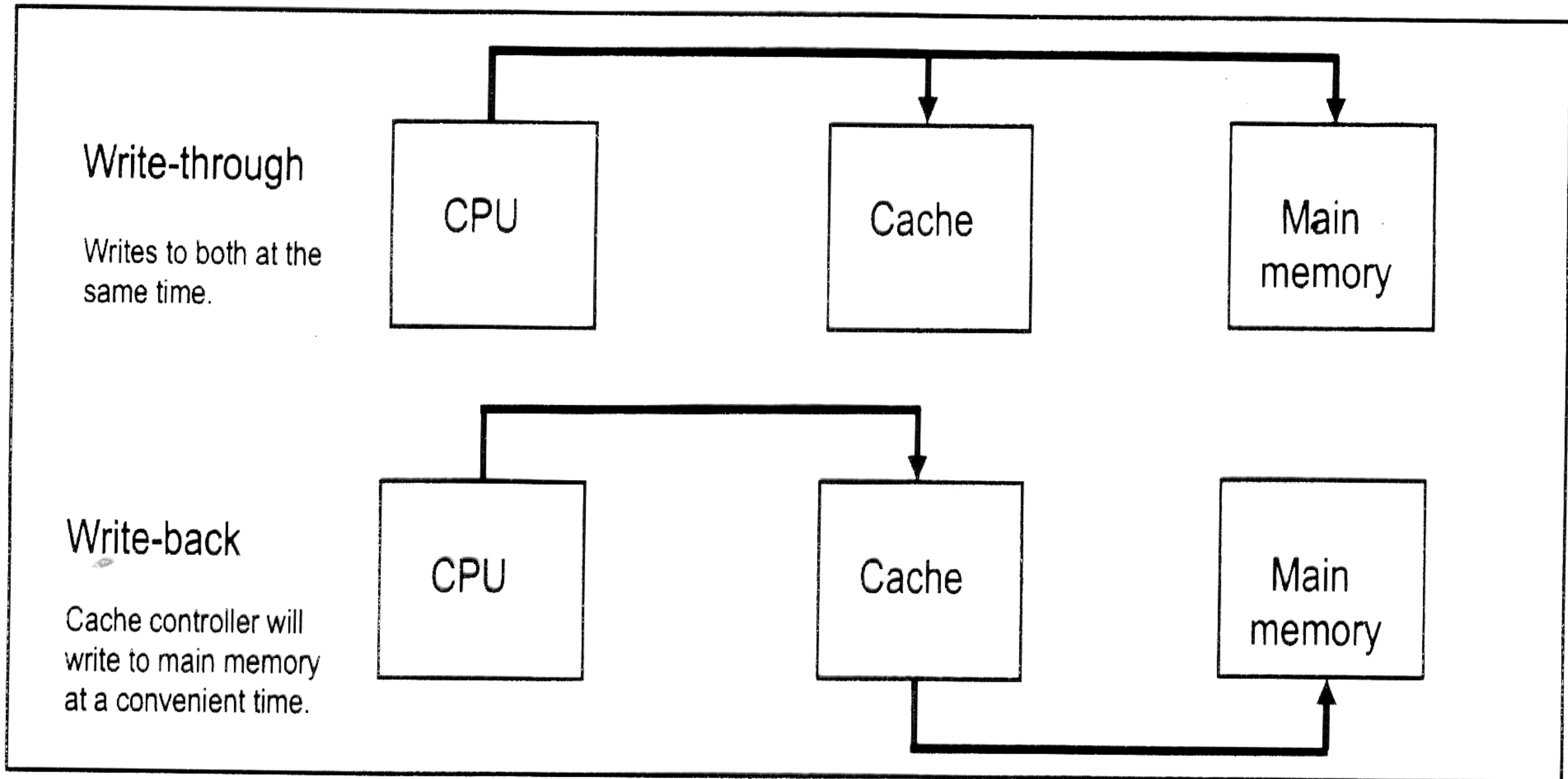


Figure 22-9. Methods of Updating Main Memory

Cache coherency

In systems in which main memory is accessed by more than one processor (DMA or multiprocessors), it must be ensured that cache always has the most recent data and is not in possession of old (or stale) data. In other words, if the data in main memory has been changed by one processor, the cache of that processor will have the copy of the latest data and the stale data in the cache memory is marked as dirty (stale) before the processor uses it. In this way, when the processor tries to use the stale data, it is informed of the situation. In cases where there is more than one processor and all share a common set of data in main memory, there must be a way to ensure that no processor uses stale data. This is called *cache coherency*.

Cache replacement policy

What happens if there is no room for the new data in cache memory and the cache controller needs to make room before it brings data in from main memory? This depends on the *cache replacement policy* adopted. In the LRU (least recently used) algorithm, the cache controller keeps account of which block of cache has been accessed (used) the least number of times, and when it needs room for the new data, this block will be swapped out to main memory or flushed if a copy of it already exists in main memory. This is similar to the relation between virtual memory and main memory. The other replacement policies are to overwrite the blocks of data in cache sequentially or randomly, or use the FIFO (first in, first out) policy. Depending on the computer's design objective and its intended use, any of these replacement policies can be adopted.

Cache fill block size

If the information asked for by the CPU is not in cache and the cache controller must bring it in from main memory, how many bytes of data are brought in whenever there is a miss? If the block size is too large (let's say 500 bytes), it will be too slow since the main memory is accessed normally with 1 or 2 WS. At the other extreme, if the block is too small, there will be too many cache misses. There must be a middle-of-the-road approach. The block size transfer from the main memory to CPU (and simultaneous copy to cache) varies in different computers, anywhere between 4 and 32 bytes. In cache controllers used with 386/486 machines, the block size is 32 bytes. This is called the *8-line cache refill policy*, where each line is 4 bytes of the 32-bit data bus. Advances in IC fabrication have allowed putting some cache on the CPU chip. This on-chip cache is called L1 (level 1) cache whereas cache outside the CPU is called L2 (level 2)

Review Questions

1. Cache is made of __________ (DRAM, SRAM).
2. From which does the CPU asks for data first, cache or main memory?
3. Rank the following from fastest to slowest as far as the CPU is concerned.
   (a) main memory (b) register (c) cache memory
4. In fully associative cache of 512 depth, there will be ____ comparisons for each data request.
5. Which cache organization requires the least number of comparisons?
6. A 4-way set associative organization requires ______ comparisons.
7. What does *write-through* refer to?
8. Which one increases the bus traffic, write-through or write-back?
9. What does LRU stand for, and how is it used?
10. A cache refill policy of 4 lines refers to ___________.

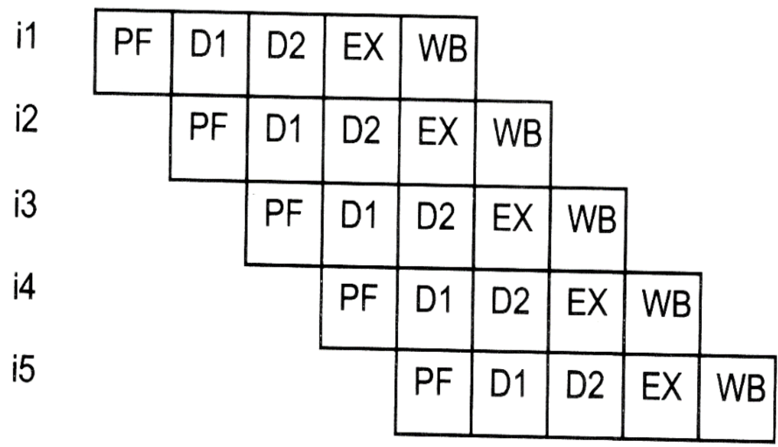
More about pipelining

In the 8085 there was no pipelining. At any given moment, it either fetched or it executed. It could not do both at the same time. In the 8085, while the buses were fetching the instructions (opcodes) and data, the CPU was sitting idle, and in the same way, when the CPU was executing instructions, buses were sitting idle. However, in the 8086/88 the fetch and execute were performed in parallel by two sections inside the CPU called the BIU (bus interface unit) and EU (execution unit). The 8086 has an internal queue where it keeps the opcodes that are prefetched and waiting for the execution unit to process them. In the sequence of instructions, if there is a jump (JMP, JNZ, JNC, and so on) or CALL, the prefetched buffer (queue) is flushed and the bus interface unit of the CPU brings in instructions from the target location while the the execution unit waits for the new instruction. Since the introduction of the 8086 in 1978, microprocessor designers have come to rely more and more on the concept of pipelining to increase the processing power of the CPU. The next development was to expand the concept of a pipeline to the three stages of fetch, decode, and execute. In the 486, the pipeline stage is broken down even further, to 5 stages as follows:

1. fetch (prefetch)
2. decode 1
3. decode 2
4. execute
5. register write-back

Due to such a large number of addressing modes in the 80x86, a two-stage decoder is used for the calculation and protection check of operand addresses. The register write-back is the stage where the operand is finally delivered to the register. For example, in the instruction "ADD EAX,[EBX+ECX*8+200]", after it is fetched, the two decoding stages are responsible for calculating the physical address of the source operand, checking for a valid address, and getting it into the CPU. There it is added together with EAX during the execution stage, and finally, the addition result is written into EAX, the destination register. Figure 23-3 shows the 486 pipeline.

This concludes the discussion of the 80486 microprocessor. For the performance comparison of the 8086, 286, 386, and 486, see Chapter 8. The performance comparison of the 386 and 486 and Pentium is shown in the next section.



PF = prefetch
D1 = decode 1
D2 = decode 2
EX = execute
WB = write back

Each stage takes 1 clock, but when the pipeline is full each instruction will execute in a single clock.

Figure 23-3. 486 Pipeline Stages

Pipelining Design Techniques

There exist two basic techniques to increase the instruction execution rate of a processor.

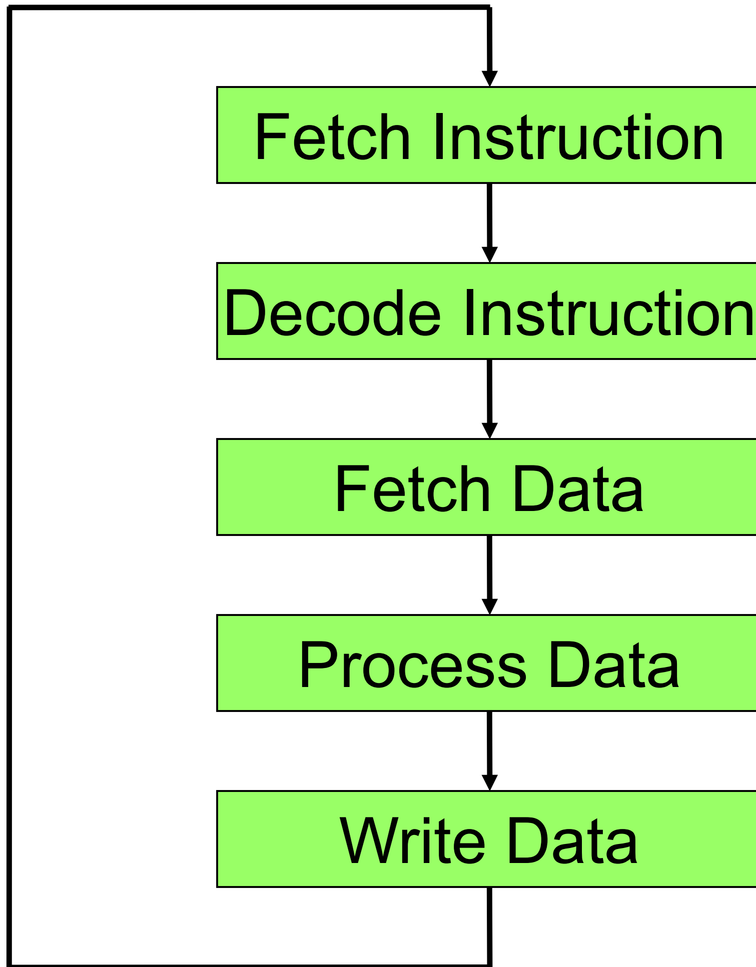
These are to increase the clock rate, thus decreasing the instruction execution time, or alternatively to increase the number of instructions that can be executed simultaneously.

Pipelining Design Techniques

- A possible division is to consider instruction fetch (F), instruction decode (D), operand fetch (F), instruction execution (E), and store of results (S) as the subtasks needed for the execution of an instruction.

- the success of a pipeline depends upon dividing the execution of an instruction among a number of subunits (stages), each performing part of the required operations.

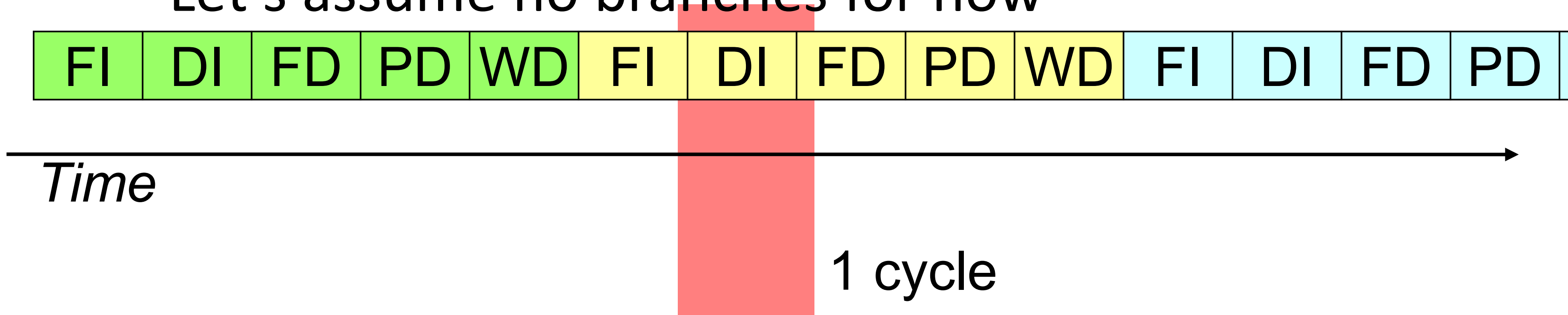
Simple Instruction Cycle



- Read the instruction from main memory
- Decode to query the requested action
- Get the data required for the requested action
- Perform the requested data processing
- Store the result of the processed data

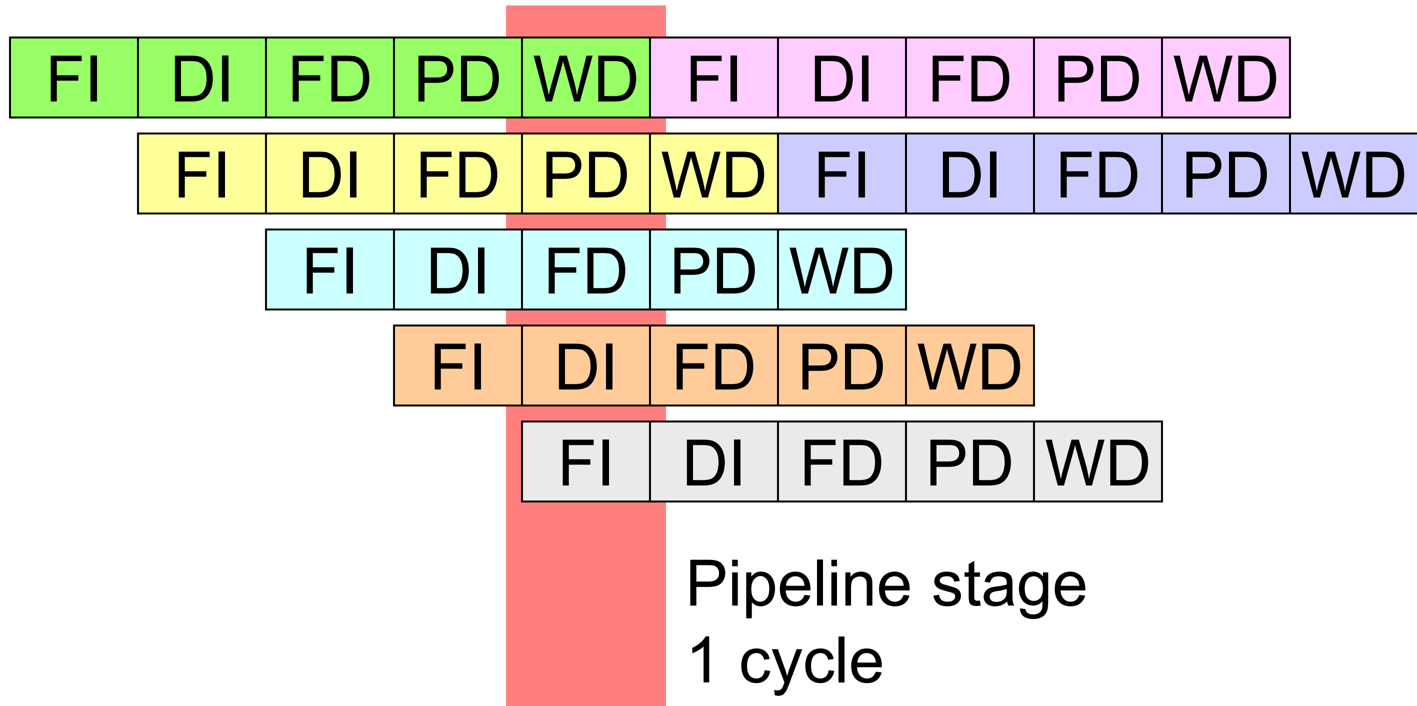
Running a Processor

- A true Stream of Instructions
  - Let's assume no branches for now



- Each step is carried out by a different unit
  - During one cycle only one unit activated
  - Others are idle

Optimization



- Make use of otherwise unused units
  - Concept is called „Pipelining"
  - Think of assembly lines