Pipeline- The modern computer architecture revolution

Seehwan Yoo

Dankook University

DKU DANKOOK UNIVERSITY

Now, you know how to execute programs inside CPU (microprocessor)

- Instructions
  - CPU-executable bit stream
  - simple primitives for complex operations
    - add, load/store, jump, branch, etc.
- Instruction Set
  - set of instructions that a computer can execute
  - different computers have different instruction sets
    - ARM instruction set != MIPS instruction set != VAX, Intel PC, etc.
- Instruction set architecture
  - instruction set determines how a computer (hardware) is composed
    - and instruction set defines how to use your computer
    - e.g.) stack machine, register numbers, basic memory/cpu operations

Complex instruction set architecture – Reduced instruction set architecture

- computers (microprocessors) can be much more complex or
  - can be much simpler than expected!
    - as you've witnessed
- Complex instructions set computer – Intel IA32/64
  - variable instruction length, predicate execution, out-of-order execution, wide-issue deep pipeline
- Reduced instructions set computer – MIPS, ARM
  - fixed-length, simple pipeline stages
- For software (backward) compatibility, ISA changes rarely
  - u-arch changes, ARMv4/v5/v6/v7/v8
  - IA-32/IA-64,
    - ISA actually evolves!

Instruction execution with hardware components

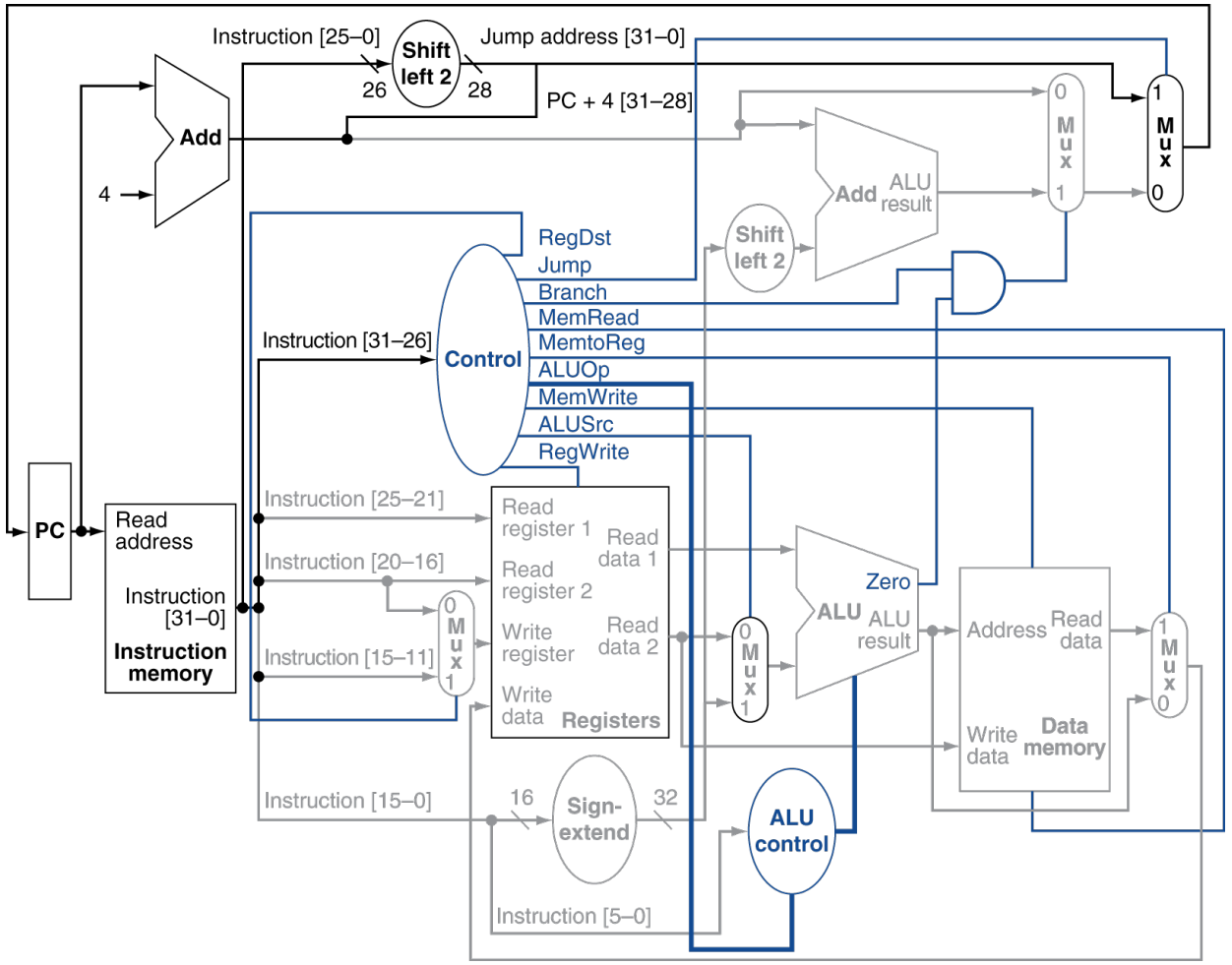
- separate data path from control path
- principles in data path design
  - critical path design
    - minimize longest logic gate path
  - bread and butter design
    - instructions are irregularly distributed/executed
      - some instructions are more frequently used (executed) than the others
    - Therefore, optimize for common (popular) cases
  - balanced design
    - there's always trade-offs
      - if you enhance something, there's disadvantages
      - e.g.) putting more hardware components vs. more instructions with simpler instruction set
      - putting single adder? two adders? or more?

Big power to change cpu design

- Moore's law
  - we can put 2x more hardware for every 18 months
    - in the same chip
    - So, we will have more hardware logic gates
    - utilize more logic gates for higher throughput!
  - → leads to ILP
    - utilize all logic gates at the same time

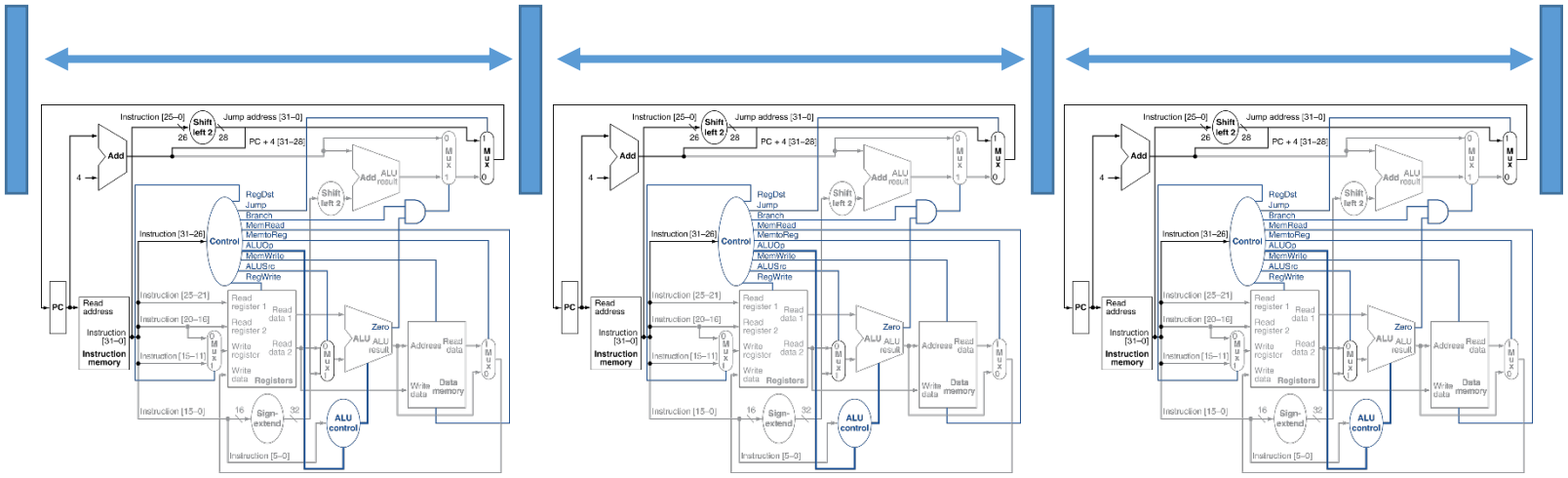
hardware design – that you understand, so far

- single cycle MIPS design
- Now, you (should) see a big picture



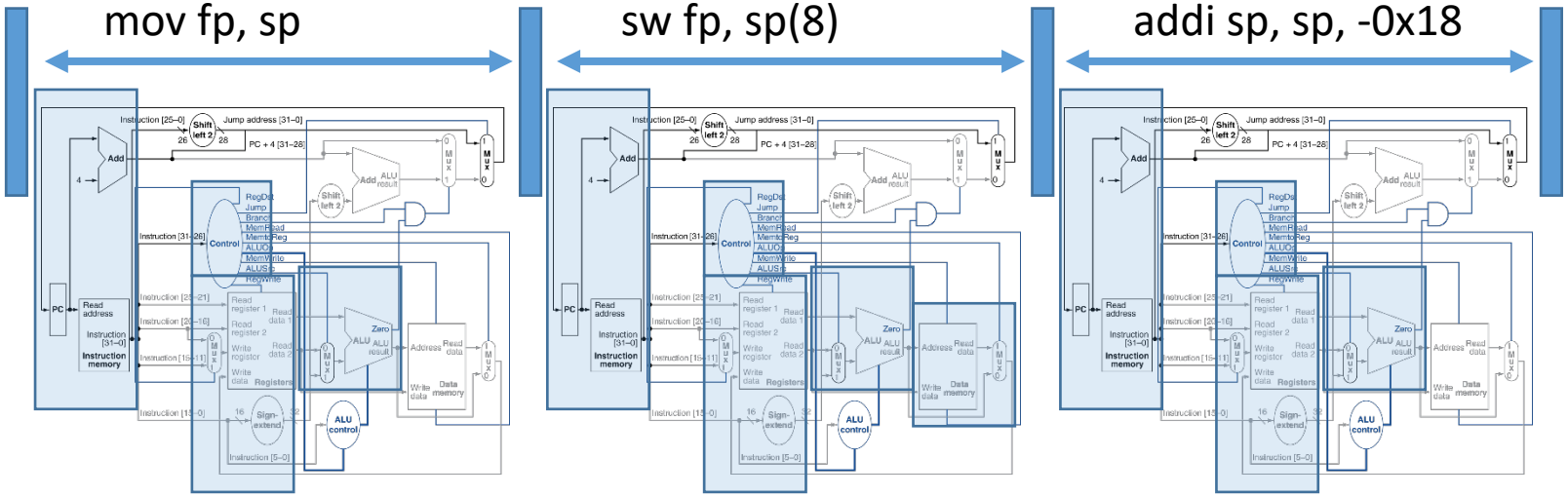
The limitation of single cycle design

- minimum cycle time:
  - time duration from fetching PC to update registers



The limitation of single cycle design

- hardware utilization
  - within a single cycle, only partial hardware components are utilized



Try more than single-cycle design!

- multi-cycle, pipelined execution
  - toward ILP

Instruction-Level Parallelism

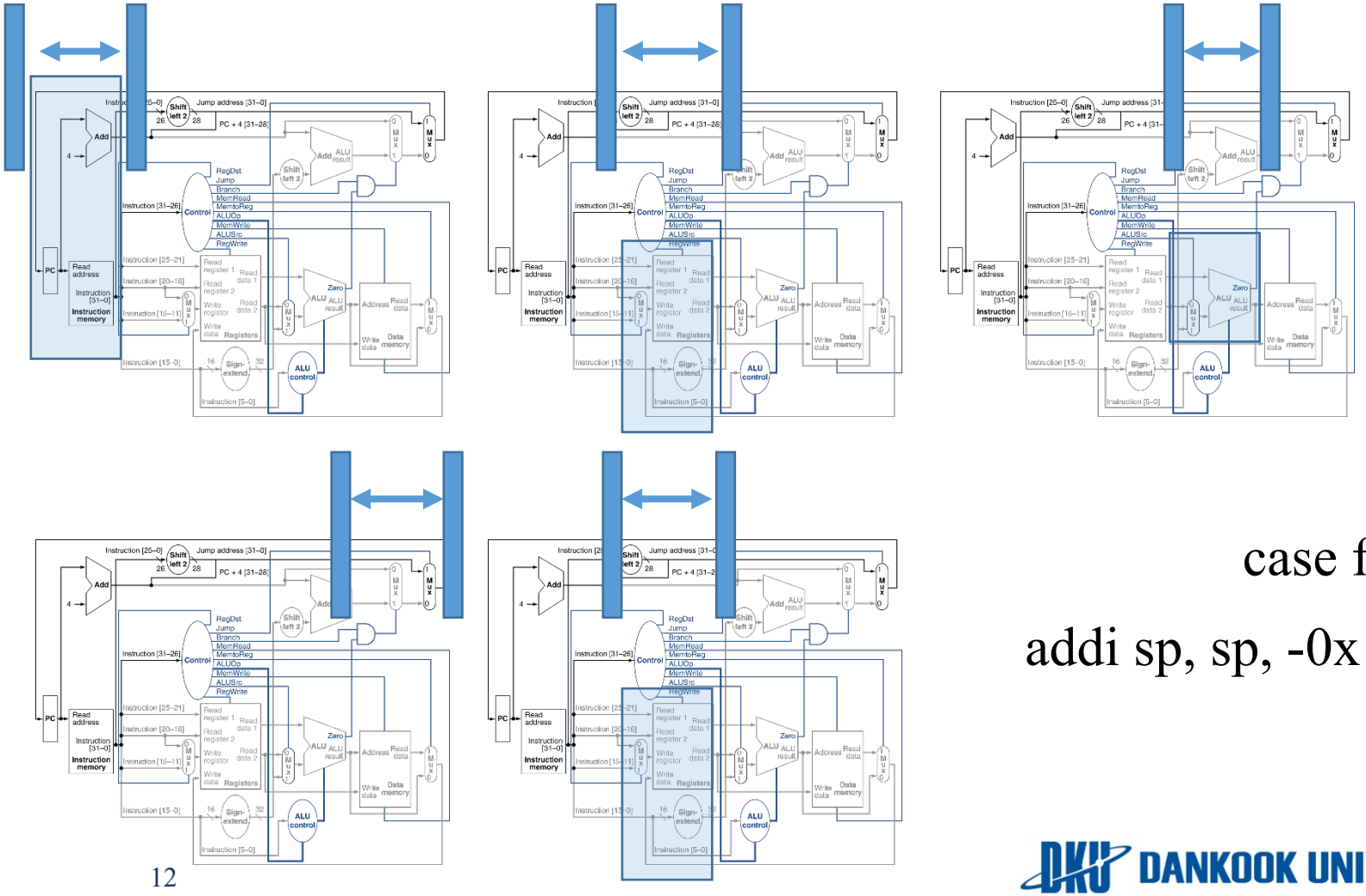
- Executing multiple instructions at the same time
  - revolutionary architecture in modern computers!
- We will look into
  - How to implement ILP? and
  - Dangling problems that comes with ILP, and
    - solutions to the problems around pipelined design

A basic idea

- divide an instruction execution into smaller steps (stages)
  - each stage for each cycle
  - + minimize cycle time
  - - now you have to remember the execution state

conceptual diagram for execution

- basic pipeline stages: IF, ID, EXE, MEM, WB

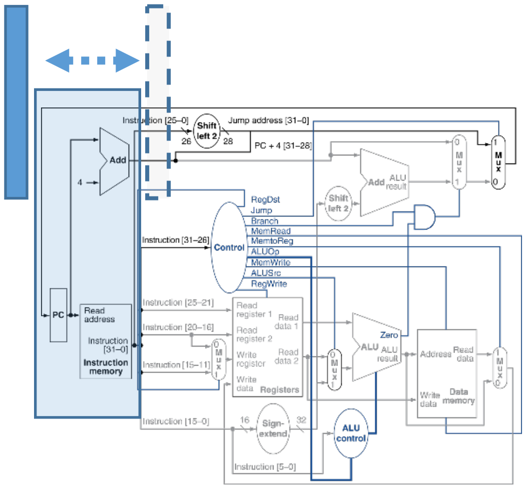


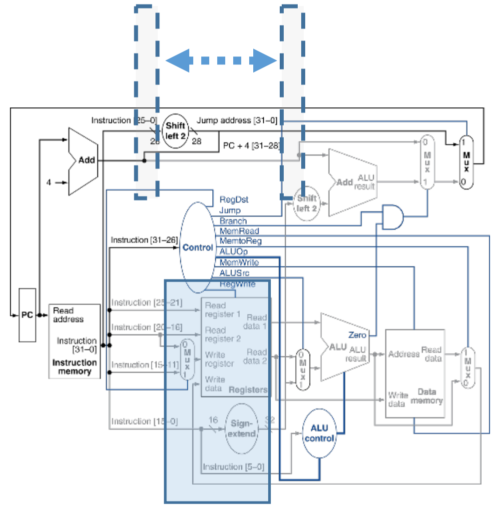
case for

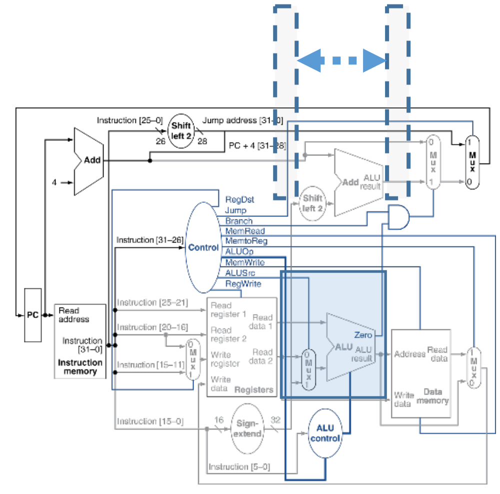
addi sp, sp, -0x18

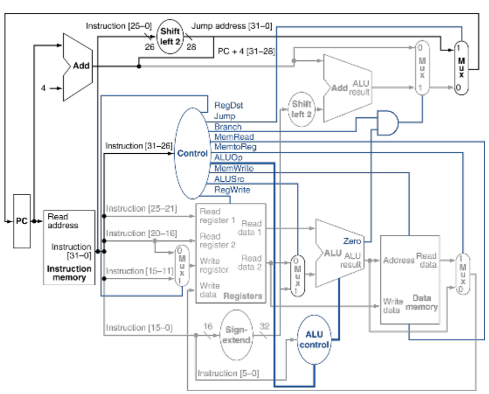
compare with single-cycle design

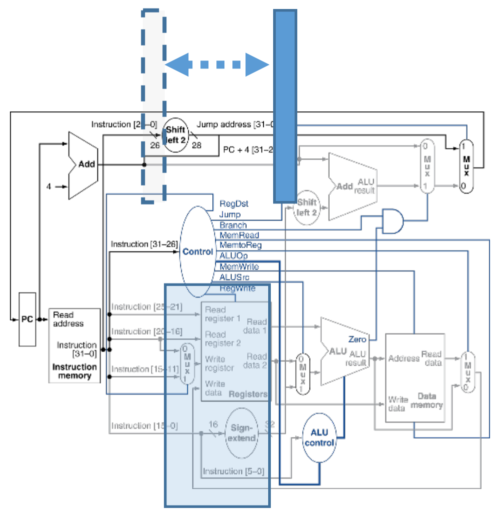
- basic pipeline stages: IF, ID, EXE, MEM, WB









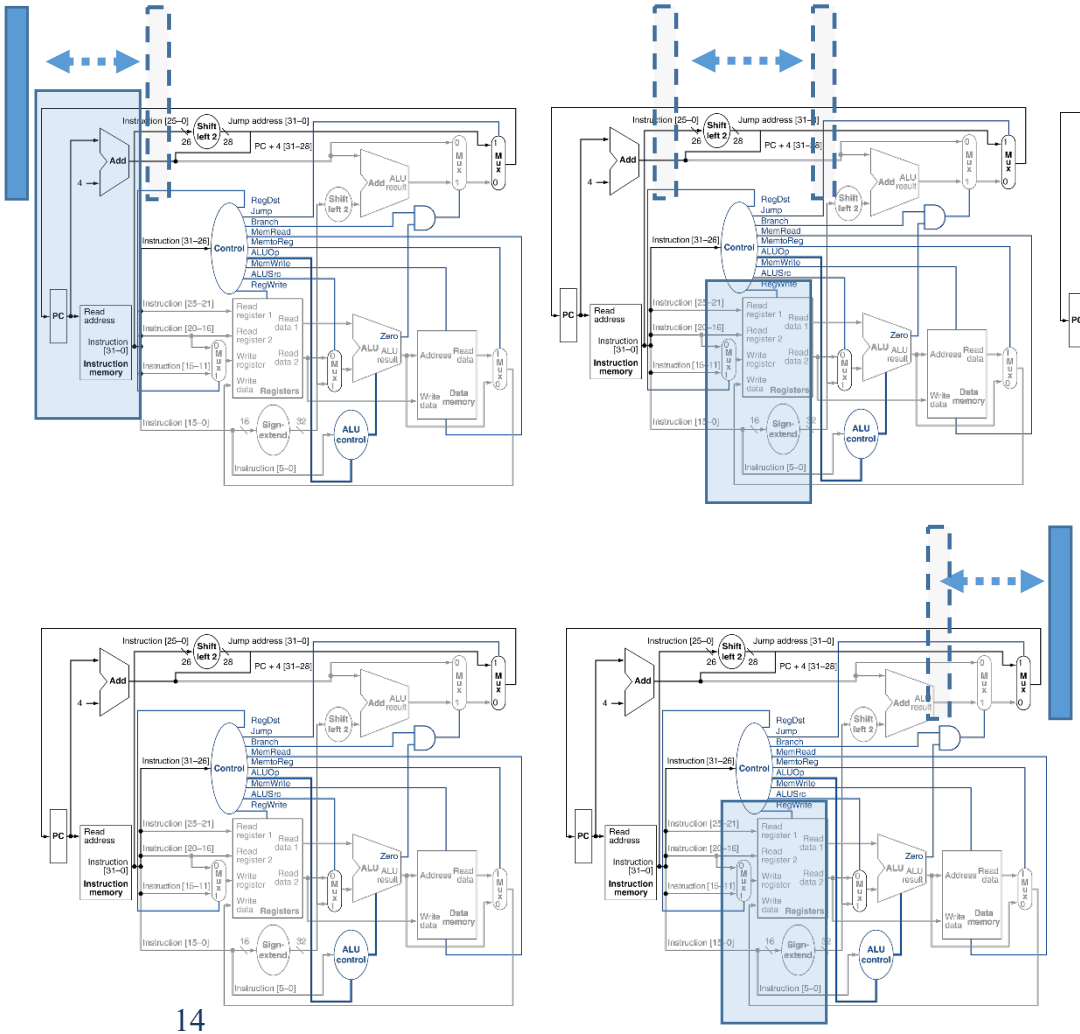


case for

addi sp, sp, -0x18

compare with single-cycle design

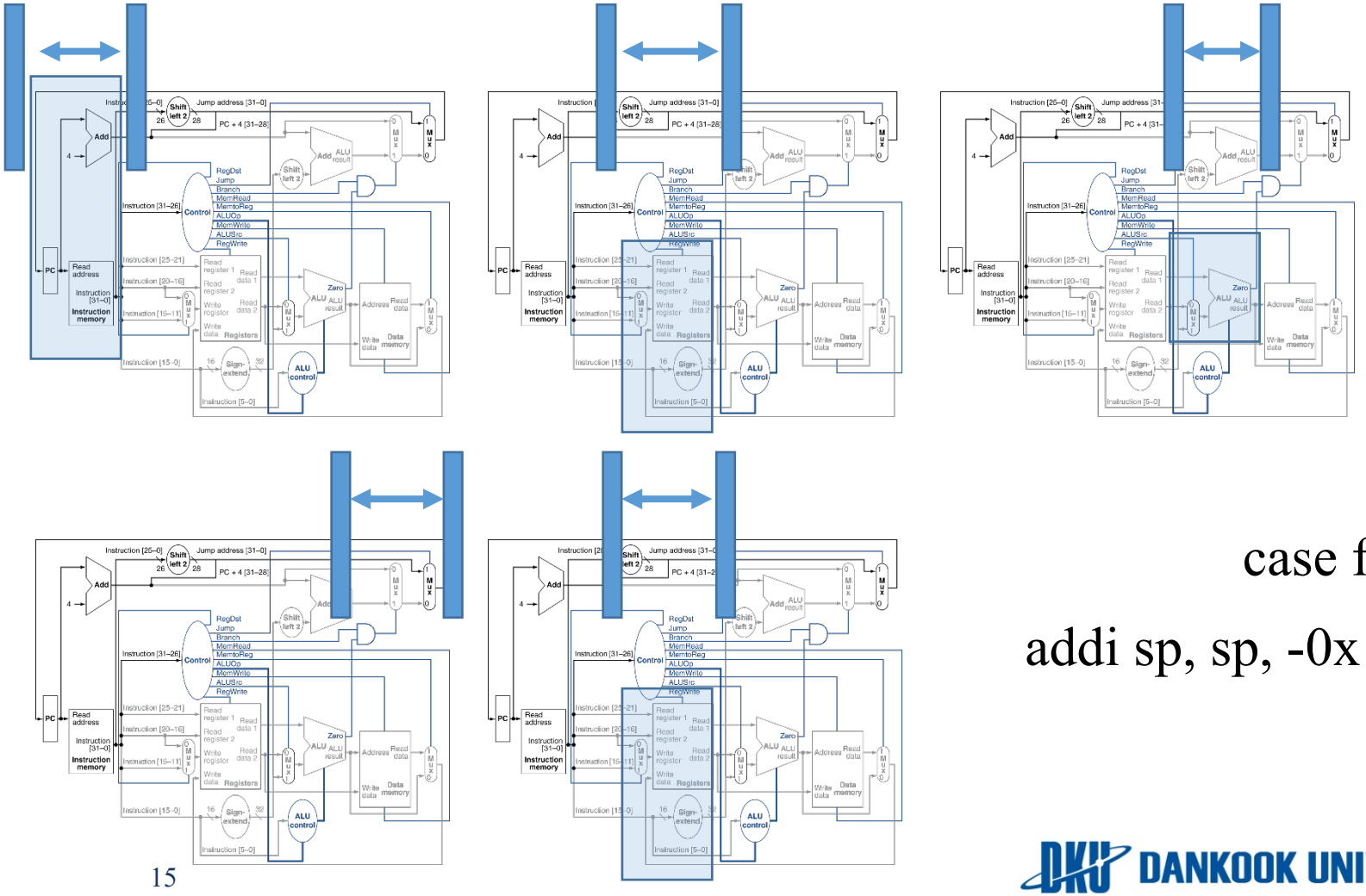
- basic pipeline stages: IF, ID, EXE, MEM, WB



DANKOOK UNIVERSITY

conceptual diagram for execution

- basic pipeline stages: IF, ID, EXE, MEM, WB

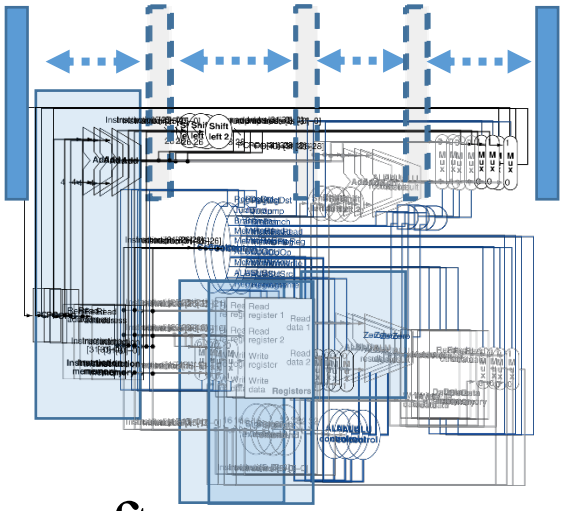


case for

addi sp, sp, -0x18

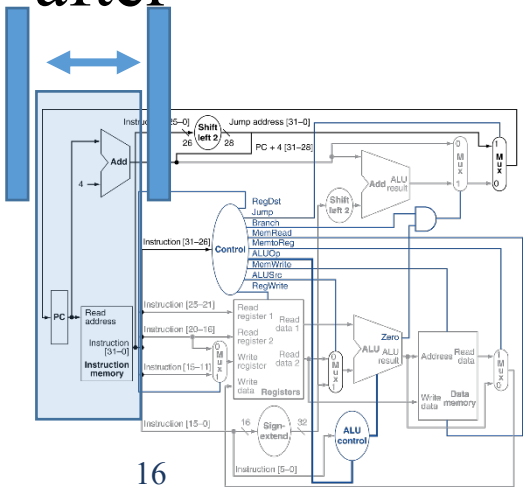
Cycle time reduction (single→ pipeline)

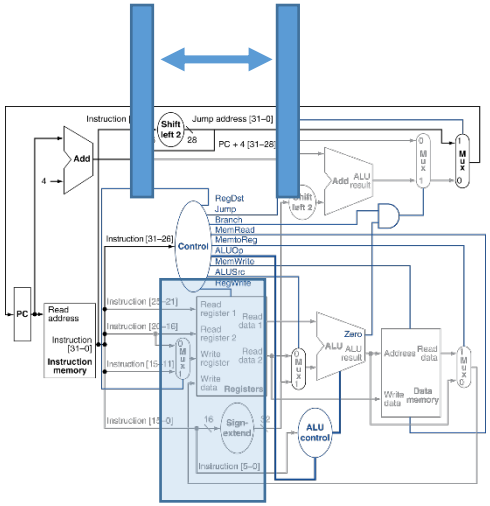
- before

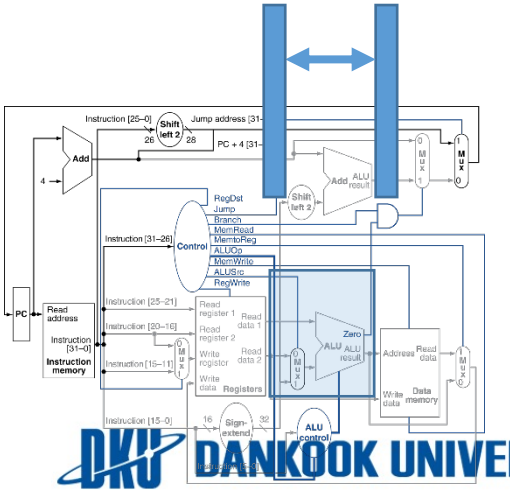


Now, we can work with ¼ cycle time

- after



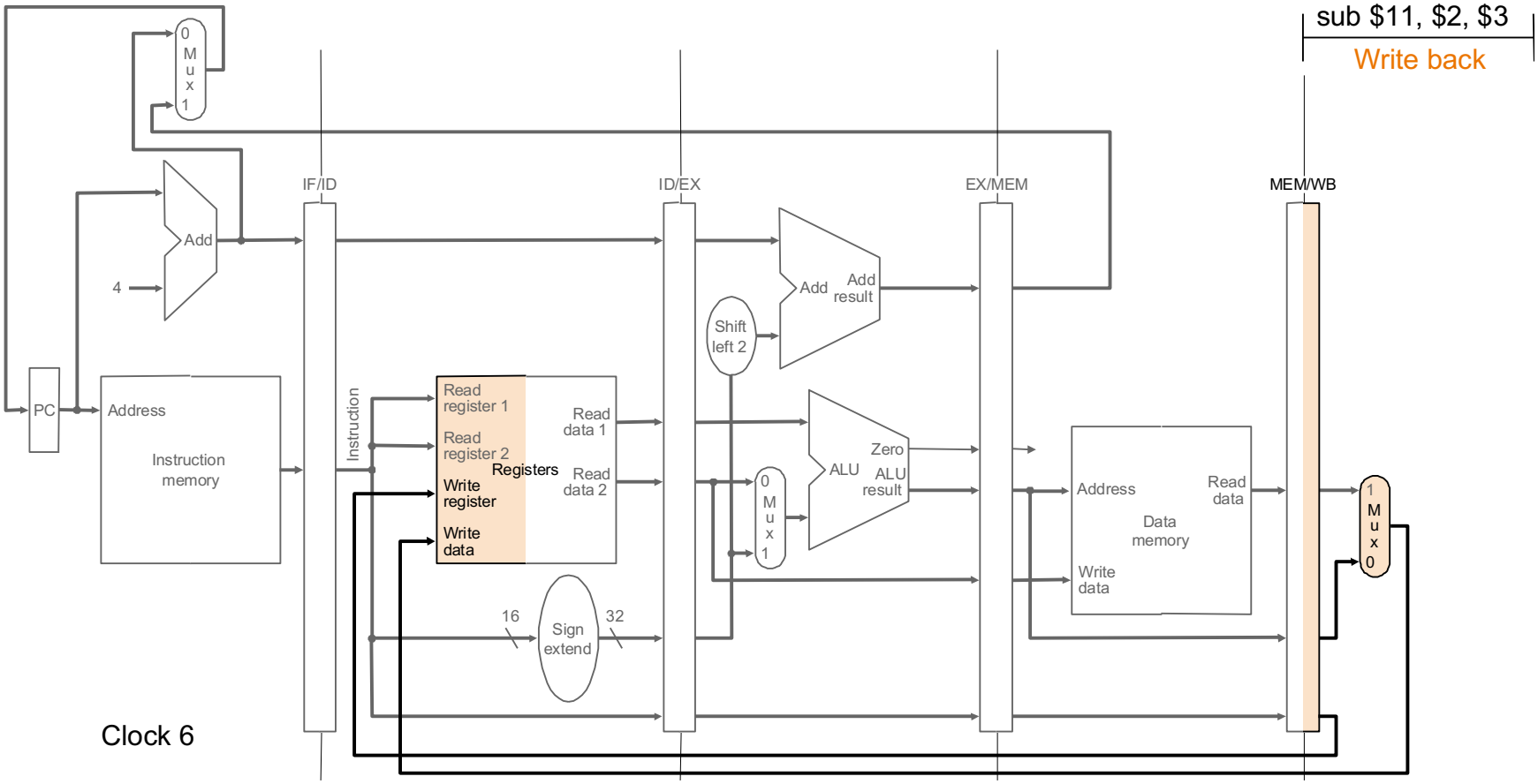




And more: overlap instructions

- In this way…

Pipelined Operation Example



Different cycle, different instructions execution

- At every clock cycle, a new instruction is fetched from memory
- At every clock cycle, each the instruction execution stage gets new and different instruction

- say there are five different R-type instructions in the pipeline
- Now you cannot use execution code in the following way
  - R[rd] = R[rs]+R[rt];
  - whose rd, rs, rt?
  - there are five different instructions in the same cycle!

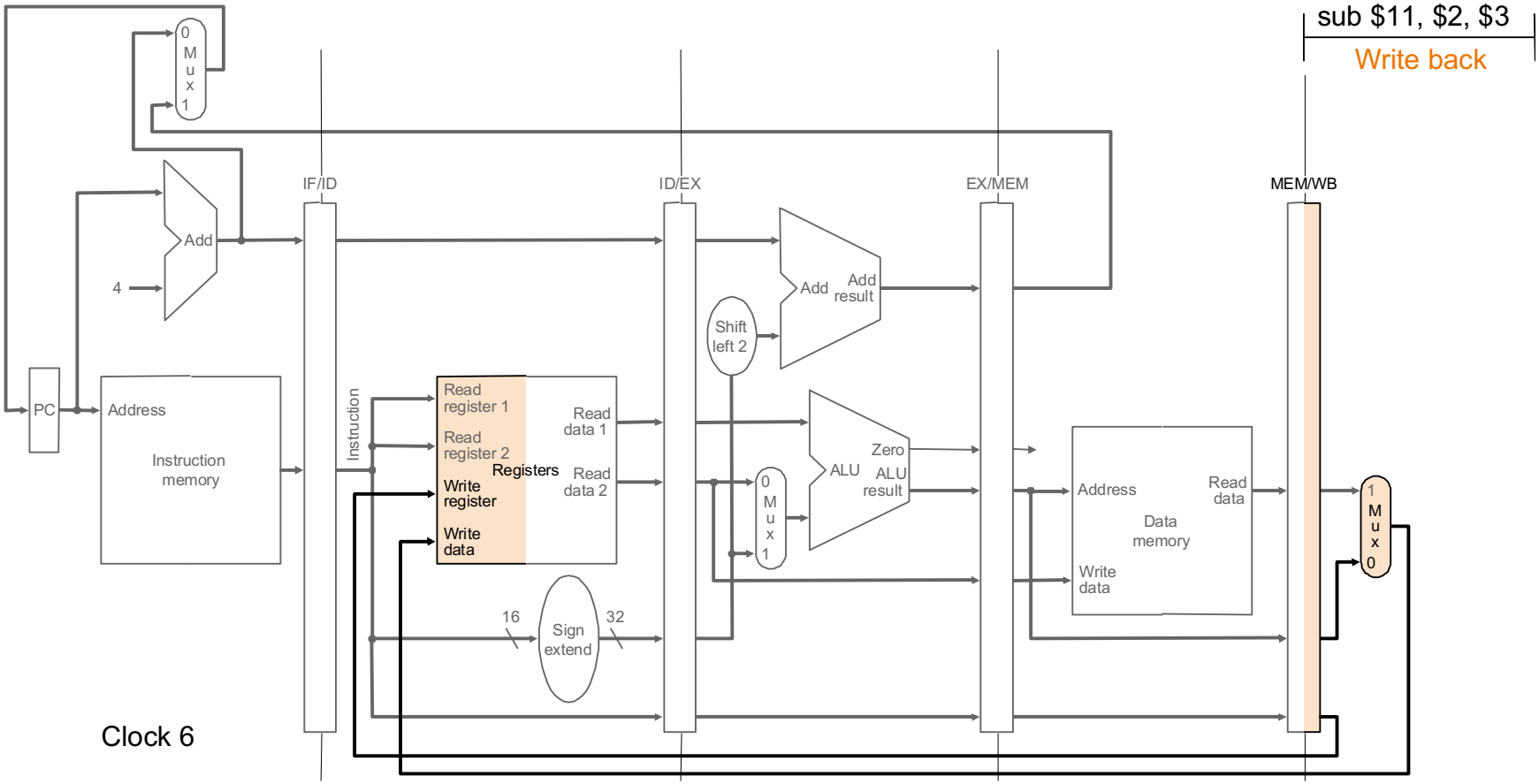
Pipeline stage has to be dependent only upon the output of the previous stage

- There is no ‘global’ rs, rd, rt!

To implement a pipelined cpu,

- you should clearly define input/output of stages
- each stage execution should be only dependent upon input data
- must produce output data for the next stage

Which stage need which input data?



Latches

- temporary storage for the state of the pipeline stages
  - hold input/output values for the next cycle
- At the beginning of the cycle, latched register values are flushed
  - updated to the new value

This is the basic pipeline structure

- enables you to execute multiple instructions in parallel
- enhances throughput
  - throughput = amount of work done / execution time
  - for 5-stage pipeline,
    - ideally 5 instructions can be executed in parallel
    - increase instructions per cycle (IPC)
      - decrease CPI (cycles per instruction)
- Remember?
  - execution time = (# of instruction) * (CPI) * (clock cycle time)

A good pipeline execution condition

- same instructions for fill in pipeline hardware
- independent instructions sequences
- identical latency for all stages

In reality…

- those conditions are rarely met…
- mostly because of the
  - resource contention
    - e.g.) memory/register access at two different stages
    - if we have single hardware (wired line), there's no way to access data at two different stages
  - and dependency
    - access registers that are used in previous pipeline stages