CS 203A
Advanced Computer Architecture

Lecture 2

Instruction Sets, Pipelining

1

RISC Vs CISC

- CISC (complex instruction set computer)
  - VAX, Intel X86, IBM 360/370, etc.
- RISC (reduced instruction set computer)
  - MIPS, DEC Alpha, SUN Sparc, IBM 801

2

RISC vs. CISC

- Characteristics of ISAs

CISC	RISC
Variable length instruction	Single word instruction
Variable format	Fixed-field decoding
Memory operands	Load/store architecture
Complex operations	Simple operations

3

RISC vs. CISC Instruction Set Design

- The historical background:
  - In first 25 years (1945-70) performance came from both technology and design.
  - Design considerations:
    - small and slow memories: compact programs are fast.
    - small no. of registers: memory operands.
    - attempts to bridge the semantic gap: model high level language features in instructions.
    - no need for portability: same vendor application, OS and hardware.
    - backward compatibility: every new ISA must carry the good and bad of all past ones.

Result: powerful and complex instructions that are rarely used.

4

Top 10 80x86 Instructions

° Rank instruction Integer Average Percent total executed

1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

° Simple instructions dominate instruction frequency

5

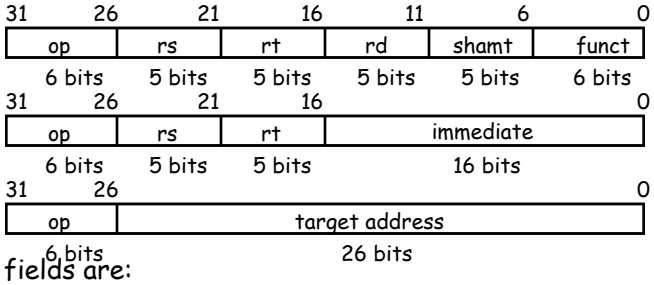
RISC vs. CISC Instruction Set Design

- Emergence of RISC
  - Very large scale integration (processor on a chip)
  - Registers – load/store ISA. Micro-store occupied about 70% of chip area: replace micro-store with registers.
  - Increased difference between CPU and memory speeds.
  - Complex instructions were not used by new compilers.
  - reduced reliance on assembly programming, new ISA can be introduced.
  - standardized vendor independent OS (Unix) became very popular in some market segments (academia and research) – need for portability
- Early RISC projects: IBM 801 (America), Berkeley SPUR, RISC I and RISC II and Stanford MIPS.

6

The MIPS Instruction Formats

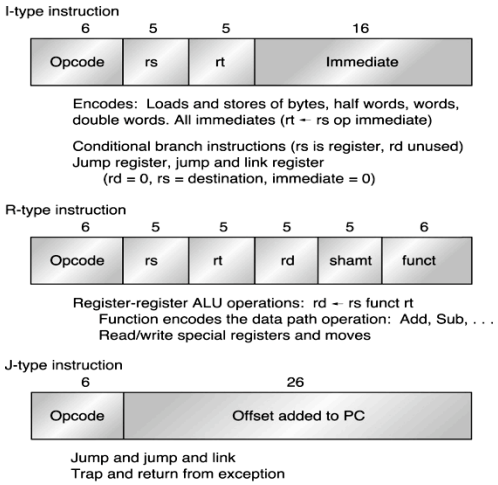
- All MIPS instructions are 32 bits long. The three instruction formats:
  - R-type
  - I-type
  - J-type



- The different fields are:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the "op" field
  - address / immediate: address offset or immediate value
  - target address: target address of the jump instruction

7

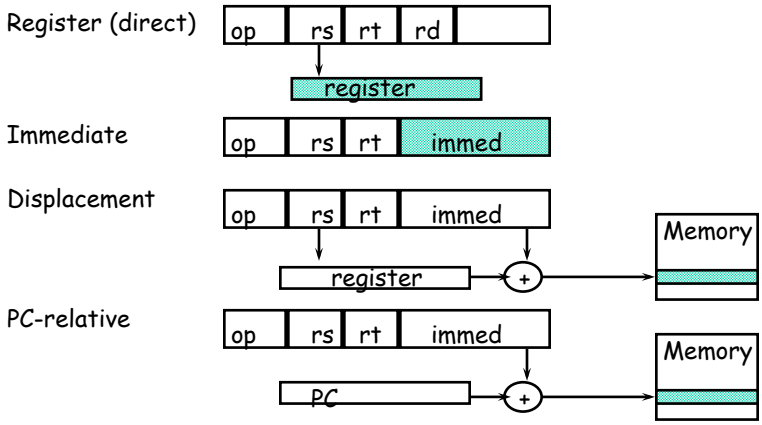
MIPS Instruction Layout



8

MIPS Addressing Modes/Instruction Formats

- All instructions 32 bits wide



9

Summary: Instruction Set Design (MIPS)

- Use general purpose registers with a load-store architecture: YES
- Provide at least 16 general purpose registers plus separate floating-point registers: 31 GPR & 32 FPR
- Support basic addressing modes: displacement (with an address offset size of 12 to 16 bits), immediate (size 8 to 16 bits), and register deferred; : YES: 16 bits for immediate, displacement (disp=0 => register deferred)
- All addressing modes apply to all data transfer instructions : YES
- Use fixed instruction encoding if interested in performance and use variable instruction encoding if interested in code size : Fixed
- Support these data sizes and types: 8-bit, 16-bit, 32-bit integers and 32-bit and 64-bit IEEE 754 floating point numbers: YES
- Support these simple instructions, since they will dominate the number of instructions executed: load, store, add, subtract, move register-register, and, shift, compare equal, compare not equal, branch (with a PC-relative address at least 8-bits long), jump, call, and return: YES
- Aim for a minimalist instruction set: YES

10

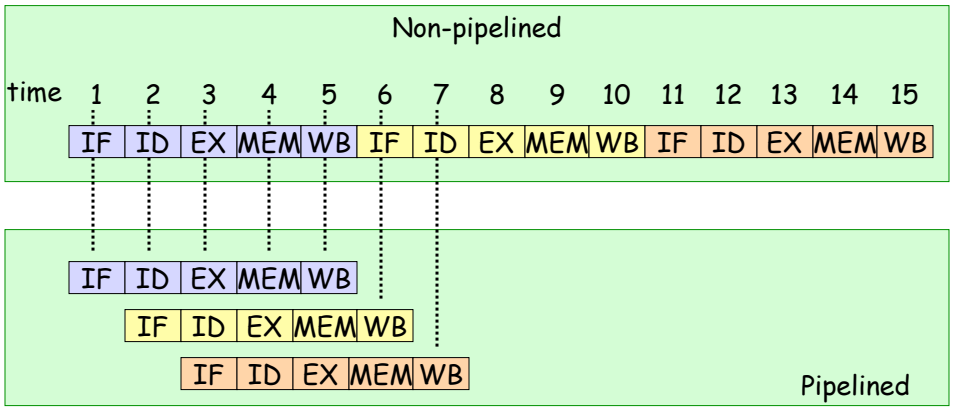
Pipelining: 5-stage Execution

- 5 stage "RISC" load-store architecture
  1. Instruction fetch (IF):
     - get instruction from memory/cache
  2. Instruction decode, Register read (ID):
     - translate opcode into control signals and read regs
  3. Execute (EX):
     - perform ALU operation, load/store address, branch outcomes
  4. Memory (MEM):
     - access memory if load/store, everyone else idle
  5. Writeback/retire (WB):
     - write results to register file

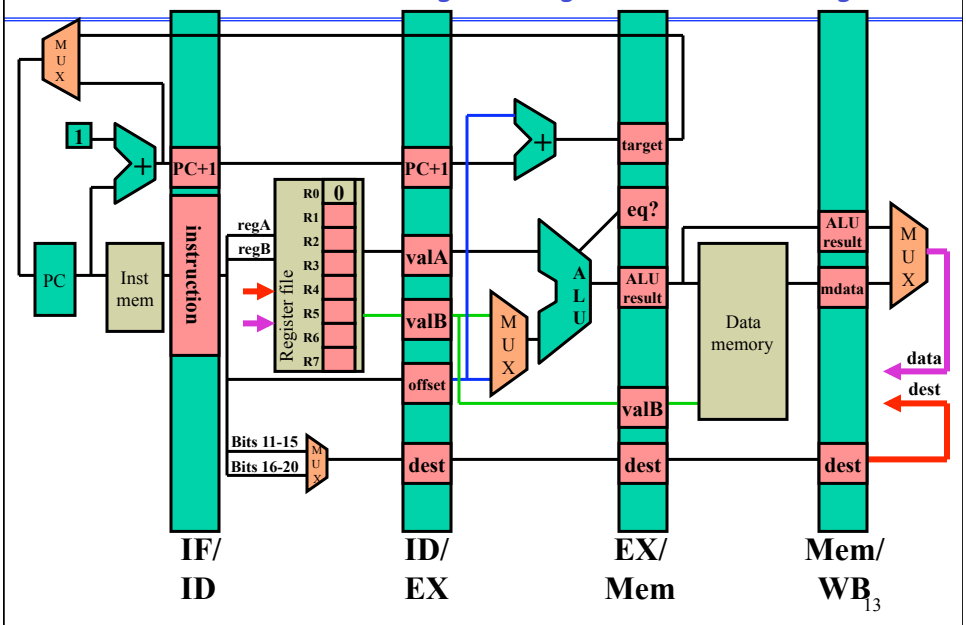
11

Solution

- Overlap execution of instructions
  - Start instruction on **every** cycle, e.g. the new instruction can be fetched while the previous one is decoded – ***pipeline***. Each cycle performing a specific task; number of stages is called pipeline depth (5 here)



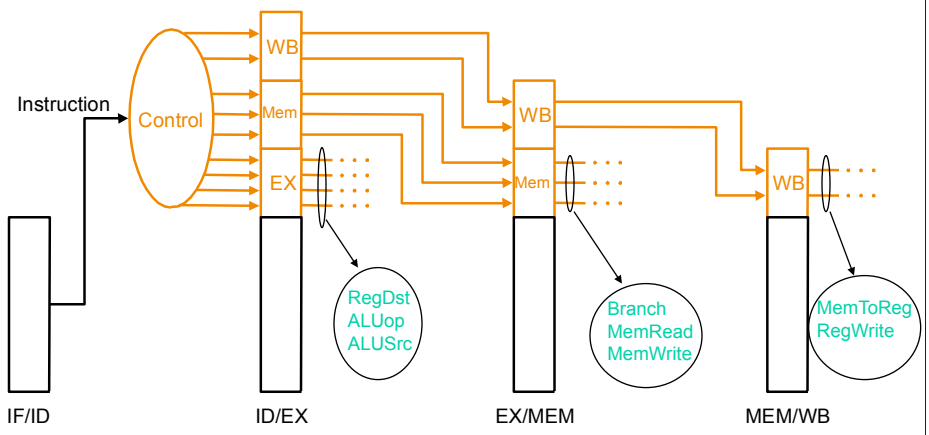
Pipeline Progress – Instn moves with all control signals, addresses, data items => different register lengths at different stages



13

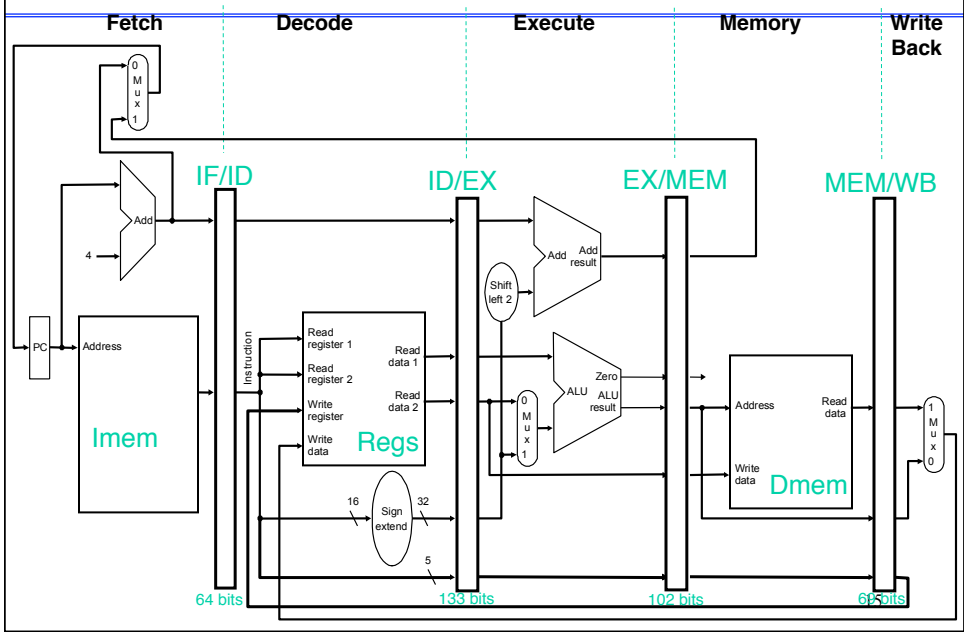
Pipelined Control

- Group control lines by pipeline stage needed
- Extend pipeline registers with control bits

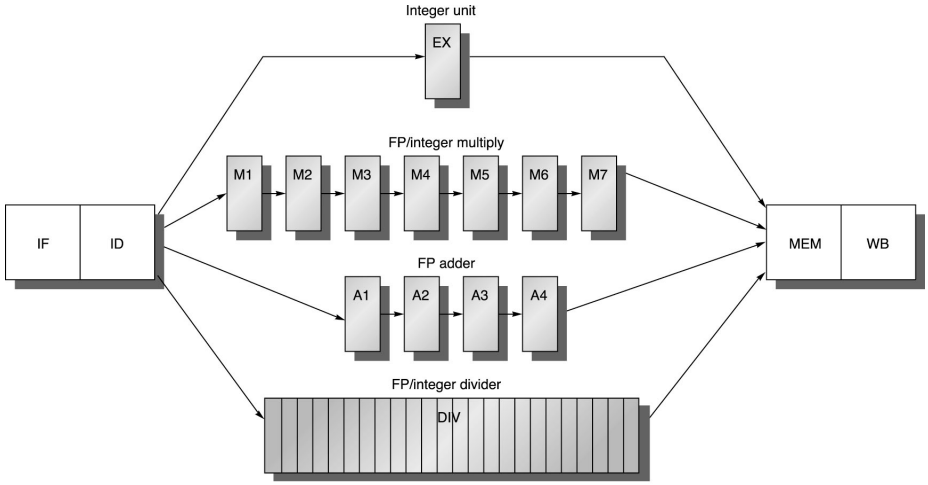


14

Pipelined Datapath (with Pipeline Regs)



A pipeline with multi-cycle FP operations: Arithmetic Pipeline: Ex. MIPS R4000



16

Pipeline Hazards

- Hazards are caused by conflicts between instructions. Will lead to incorrect behavior if not fixed.
  - Three types:
    - Structural: two instructions use same h/w in the same cycle – resource conflicts (e.g. one memory port, unpipelined divider etc).
    - Data: two instructions use same data storage (register/memory) – dependent instructions.
    - Control: one instruction affects which instruction is next – PC modifying instruction, changes control flow of program.

17

Handling Hazards

- Force stalls or bubbles in the pipeline.
  - Stop some younger instructions in the stage when hazard happens
  - Make younger instr. Wait for older ones to complete
  - Implementation: de-assert write-enable signals to pipeline registers
- Flush pipeline
  - Blow instructions out of the pipeline
  - Refetch new instructions later – solving control hazards
  - Implementation: assert clear signals on pipeline registers

18