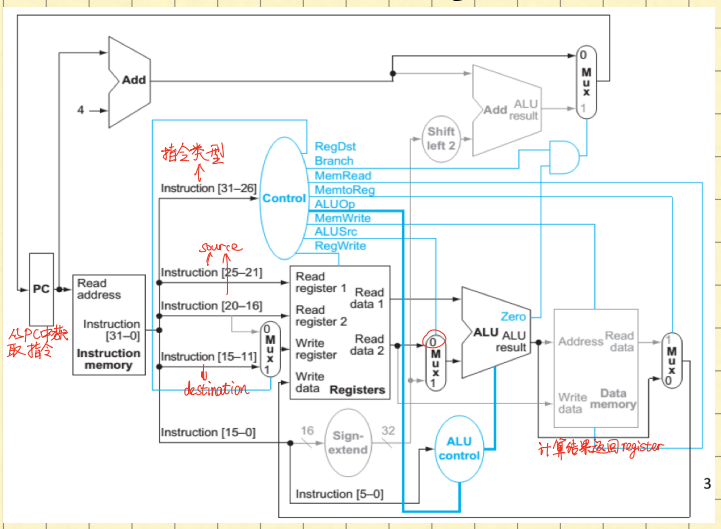
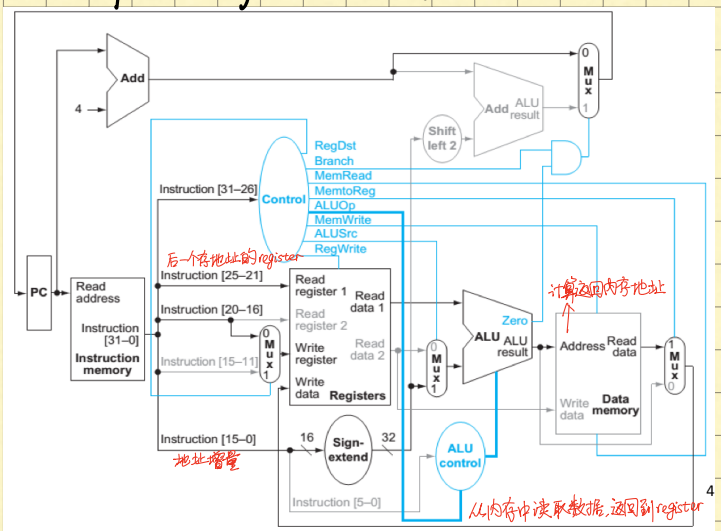
Datapath for an R-type Instruction



Datapath for a load Instruction

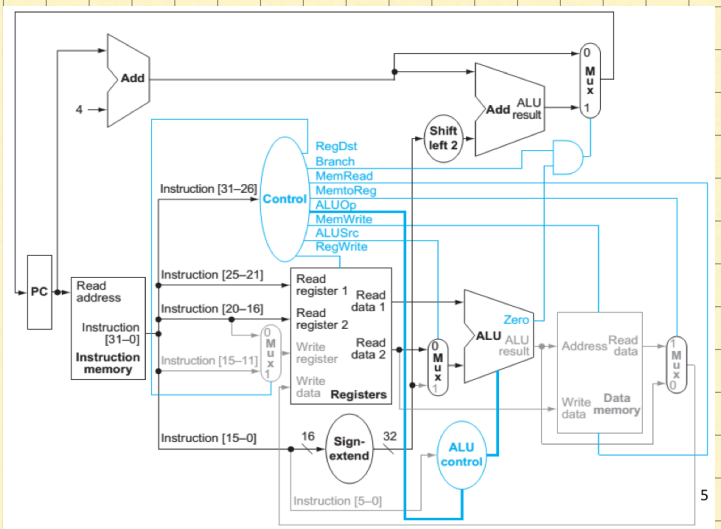
lw $s0, 4($s1)



Datapath for a Branch-on-equal Instruction

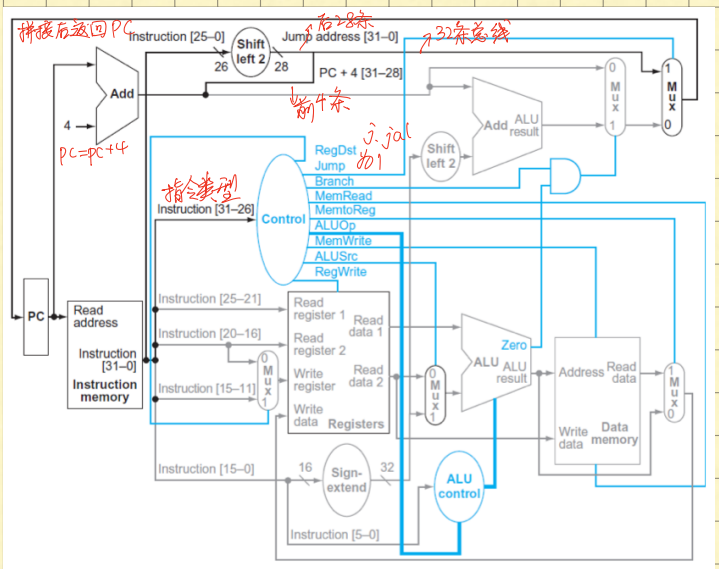
beq $s0, $s1, Label

pc-relative



Datapath for Jump

j Label pseudo-direct addressing



Truth Table for Control Unit

用组合逻辑生成这幅图

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Performance Issues

- Longest delay determines clock period
  - Critical path: load instruction
  - Instruction memory → register file → ALU → data memory → register file
- Not feasible to vary period for different instructions
- Violates design principle
  - Making the common case fast
- We will improve performance by pipelining

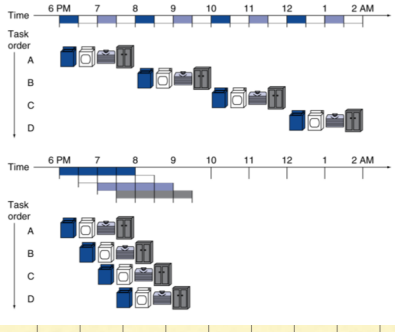
执行用时:
lw > add > beq > j

执行用时最长的指令会决定时钟周期
这违反了最常用的指令应该越短时间.

可用流水线提升.

Pipelining Analogy

- Pipelined laundry: overlapping execution
  - Parallelism improves performance



- Four loads:
  - Speedup = 16/7 = 2.3
- Non-stop:
  - Speedup = 4n/(n + 3) ≈ 4 = number of stages

重叠了执行步骤

有限条时有一些会置空

无限条指令就能达到4倍的speedup
①有限 ②Hazard ③不均匀
这三个因素会阻止到达4倍.

MIPS Pipeline

- Pipeline: an implementation technique in which multiple instructions are overlapped in execution.
- Five stages, one step per stage
  1. IF: Instruction fetch from memory
  2. ID: Instruction decode & register read
  3. EX: Execute operation or calculate address
  4. MEM: Access memory operand
  5. WB: Write result back to register

指令一般能分成5步：
①IF：获取指令
②ID：指令译码&读寄存器
③EX：执行/计算地址
④MEM：访问内存
⑤WB：写回数据到寄存器

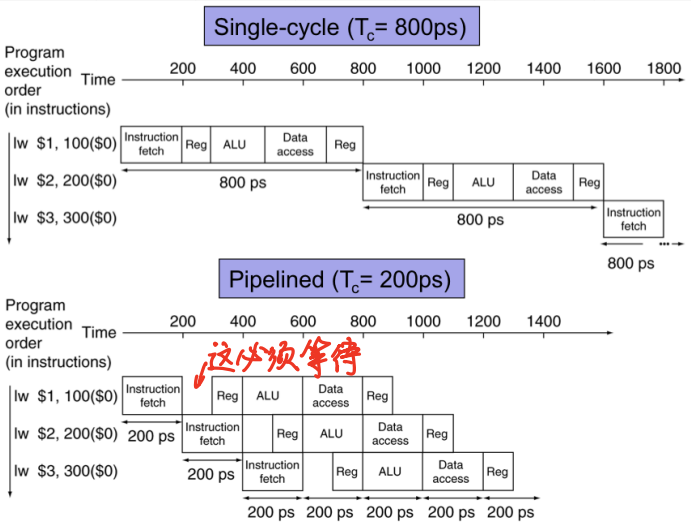
有的指令可以不需要5步.

Pipeline Performance

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

尽量让各个阶段等长，方便流水线化.



时钟周期必须满足最慢的指令.

Pipeline Speedup 流水线加速

- If all stages are balanced
  - i.e., all take the same time
  - Time between instructions$_{pipelined}$ = $\frac{\text{Time between instructions}_{nonpipelined}}{\text{Number of stages}}$
- If not balanced, speedup is less
- Speedup due to increased throughput
  - Latency (time for each instruction) does not decrease

①指令是否均匀(每个stage用时是否同)
②是否同时

每条指令用时未变短，但总体变短了.

Pipelining and ISA Design

- MIPS ISA designed for pipelining
  - All instructions are 32-bits
    - Easier to fetch and decode in one cycle
    - c.f. x86: 1- to 17-byte instructions
  - Few and regular instruction formats
    - Can decode and read registers in one step
  - Load/store addressing
    - Can calculate address in 3rd stage, access memory in 4th stage
  - Alignment of memory operands
    - Memory access takes only one cycle

从cache是1 cycle
但从真正的memory可能上百cycle.

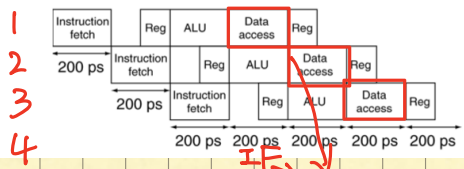
Hazards 竞争、冒险

- Situations that prevent starting the next instruction in the next cycle
- Structure hazards
  - A required resource is busy
- Data hazards
  - Need to wait for previous instruction to complete its data read/write
- Control hazards
  - Decisions of control action depends on the previous instruction

理想中每过一个clock cycle进一条指令可能会因Hazard无法实现.

Structure Hazards 结构冲突/资源冲突

- Conflict for use of a resource
- If MIPS pipeline has only one memory (data and instructions all in one), then
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
    - Would cause a pipeline "bubble"
- Hence, pipelined datapaths require separate instruction/data memories
  - Or separate instruction/data caches



pipeline stall / bubble 指是并行时需停止一段时间.

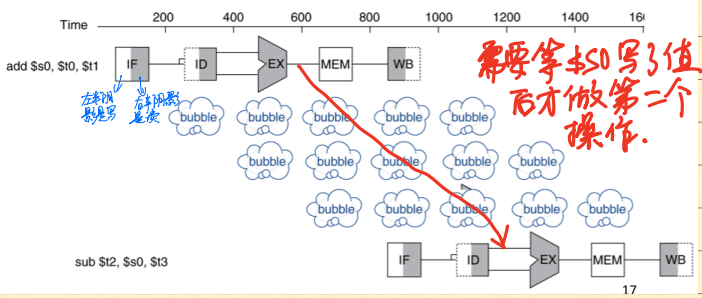
同时访问一个资源,冲突!
因此,流水线的数据通路要分开指令和数据内存.

若data和instruction同在memory中,则会发生structure Hazard.
为解决这一问题,常将data和instruction分开存.

Data Hazards 数据冲突

- An instruction depends on completion of data access by a previous instruction

```
add  $s0, $t0, $t1
sub  $t2, $s0, $t3
```

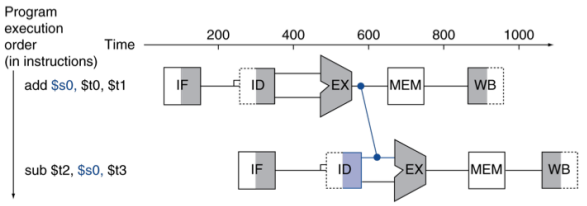


前后数据有依赖关系
可用forwarding解决.

需要等$s0写入值后才做第二个操作.

Forwarding (aka. Bypassing)

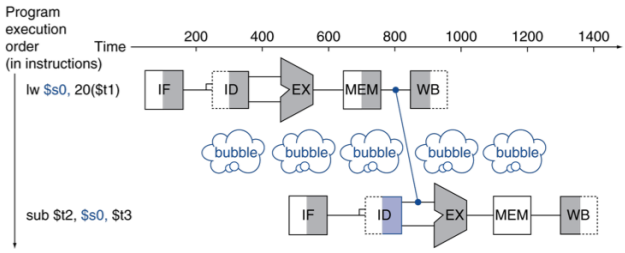
- Forwarding can help to solve data hazard
- Core idea: Use result immediately when it is computed
  - Don't wait for it to be stored in a register
  - Requires extra connections in the datapath
  - Add a bypassing line to connect the output of EX to the input



加旁路.

Load-Use Data Hazard

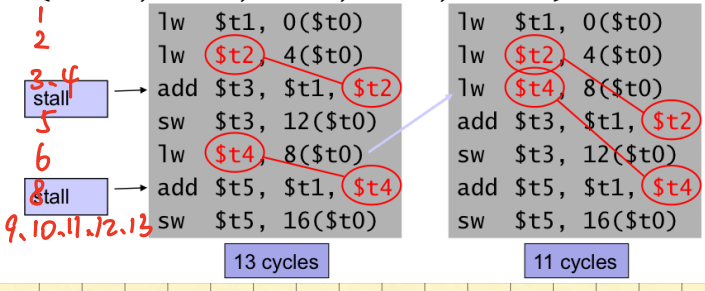
- Can't always avoid stalls by forwarding
  - If value not computed when needed
  - Can't forward backward in time!
  - E.g. lw $s0, 20($t1), sub $t2, $s0, $t3



硬件上解决，加一条旁路。
这条旁路是否用是看Compiler的分析.

Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction (avoid "load + exe" pattern)
- C code for `A = B + E; C = B + F;`
- `(t1: B, t2:E, t3:A, t4:F, t5: C)`



通过改变执行顺序，减少依赖关系.

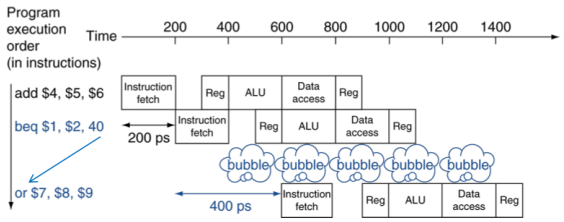
Control Hazards (Branch Hazards)

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction
    - Still working on ID stage of branch
- In MIPS pipeline
  - Need to compare registers and compute target early in the pipeline
  - Add hardware to do it in ID stage

分支跳转时不知道 bubble 要加多少个.

Stall on Branch

- Wait until branch outcome determined before fetching next instruction



Branch Prediction

- Longer pipelines can't readily determine branch outcome early
  - Stall penalty becomes unacceptable
- Predict outcome of branch
  - Stall only if prediction is wrong
- In MIPS pipeline
  - Can simply predict that branches are not taken
  - Fetch instruction after branch, with no delay

一般情况：0 bubble

if 0 then A else B

分支预测：0 A → 直接先取A

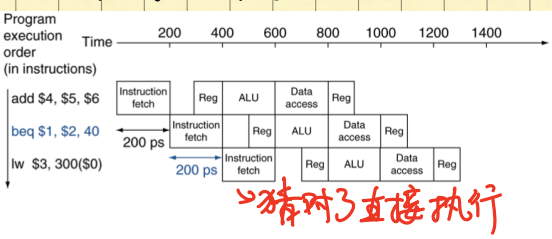
B → 若A取错了再取B

有50%预测对.

对for、while之类的循环，在达到最终结束前都可以为是无条件跳回，因此此时更容易预测对.

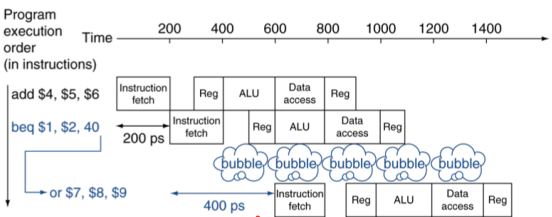
MIPS with Predict Not Taken

Prediction correct



猜对了直接执行

Prediction incorrect



猜错了会浪费一个时钟周期

More-Realistic Branch Prediction

- Static branch prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Predict backward branches taken
    - Predict forward branches not taken
- Dynamic branch prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - When wrong, stall while re-fetching, and update history

静态分支预测

动态分支预测

在执行过程中动态确认、

if 0 then A else B

会统计在执行过程中A或B的执行次数，下次会根据哪个执行更多而预先跳哪个，需为一个硬件用于统计.

Pipeline Summary

- Pipelining improves performance by increasing instruction throughput
  - Executes multiple instructions in parallel
  - Each instruction has the same latency
- Subject to hazards
  - Structure, data, control
- Instruction set design affects complexity of pipeline implementation

指令集的设计也影响pipeline的实现.