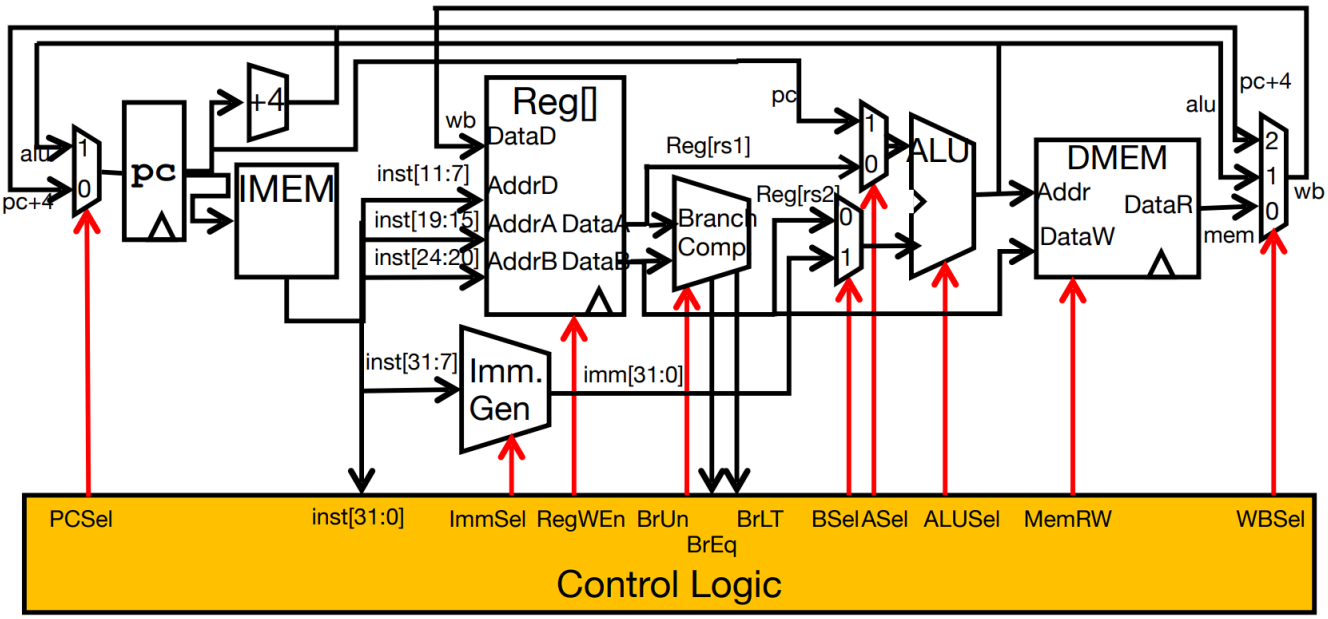
Controller



Control Logic "Truth Table" (incomplete)

Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
add	*	*	+4	-	-	Reg	Reg	Add	Read	1	ALU
sub	*	*	+4	-	-	Reg	Reg	Sub	Read	1	ALU
(R-R Op)	*	*	+4	-	-	Reg	Reg	*(Op)*	Read	1	ALU
addi	*	*	+4	I	-	Reg	Imm	Add	Read	1	ALU
lw	*	*	+4	I	-	Reg	Imm	Add	Read	1	Mem
sw	*	*	+4	S	-	Reg	Imm	Add	Write	0	-
beq	0	*	+4	B	-	PC	Imm	Add	Read	0	-
beq	1	*	ALU	B	-	PC	Imm	Add	Read	0	-
bne	0	*	ALU	B	-	PC	Imm	Add	Read	0	-
bne	1	*	+4	B	-	PC	Imm	Add	Read	0	-
blt	*	1	ALU	B	0	PC	Imm	Add	Read	0	-
bltu	*	1	ALU	B	1	PC	Imm	Add	Read	0	-
jalr	*	*	ALU	I	-	Reg	Imm	Add	Read	1	PC+4
jal	*	*	ALU	J	-	PC	Imm	Add	Read	1	PC+4
auipc	*	*	+4	U	-	PC	Imm	Add	Read	1	ALU

* means "for all values"
- means "don't care, use any value"

Note: Instruction type encoded using only 9 bits inst[30],inst[14:12], inst[6:2]

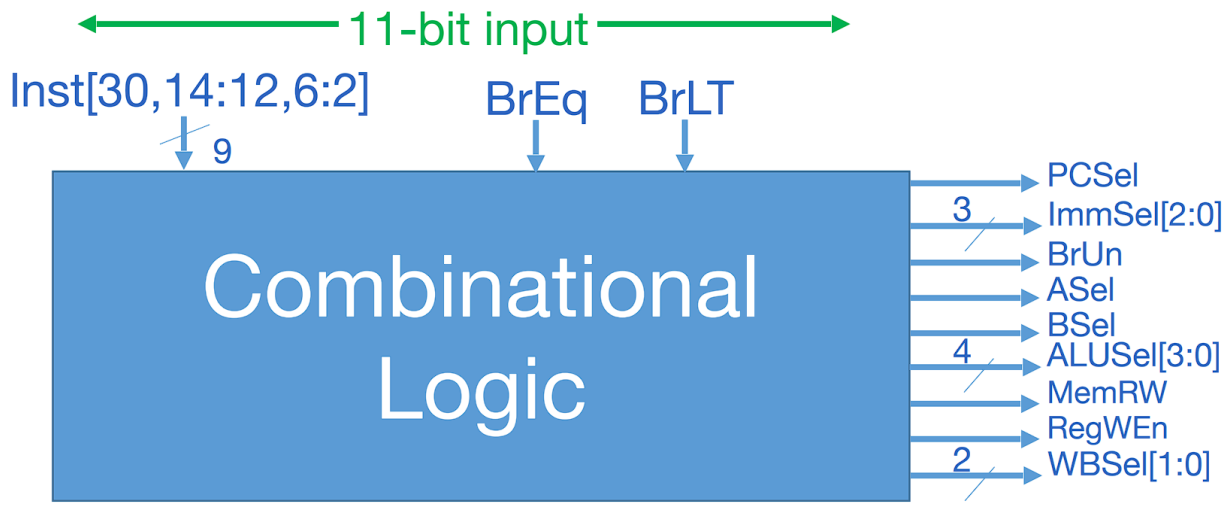
imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI

inst[30] inst[14:12] inst[6:2]

0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
0000	pred	succ	00000	000	00000	0001111	FENCE
0000	0000	0000	00000	001	00000	0001111	FENCE.I
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK
csr			rs1	001	rd	1110011	CSRRW
csr			rs1	010	rd	1110011	CSRRS
csr			rs1	011	rd	1110011	CSRRC
csr			zimm	101	rd	1110011	CSRRWI
csr			zimm	110	rd	1110011	CSRRSI
csr			zimm	111	rd	1110011	CSRRCI

Not in CS61C

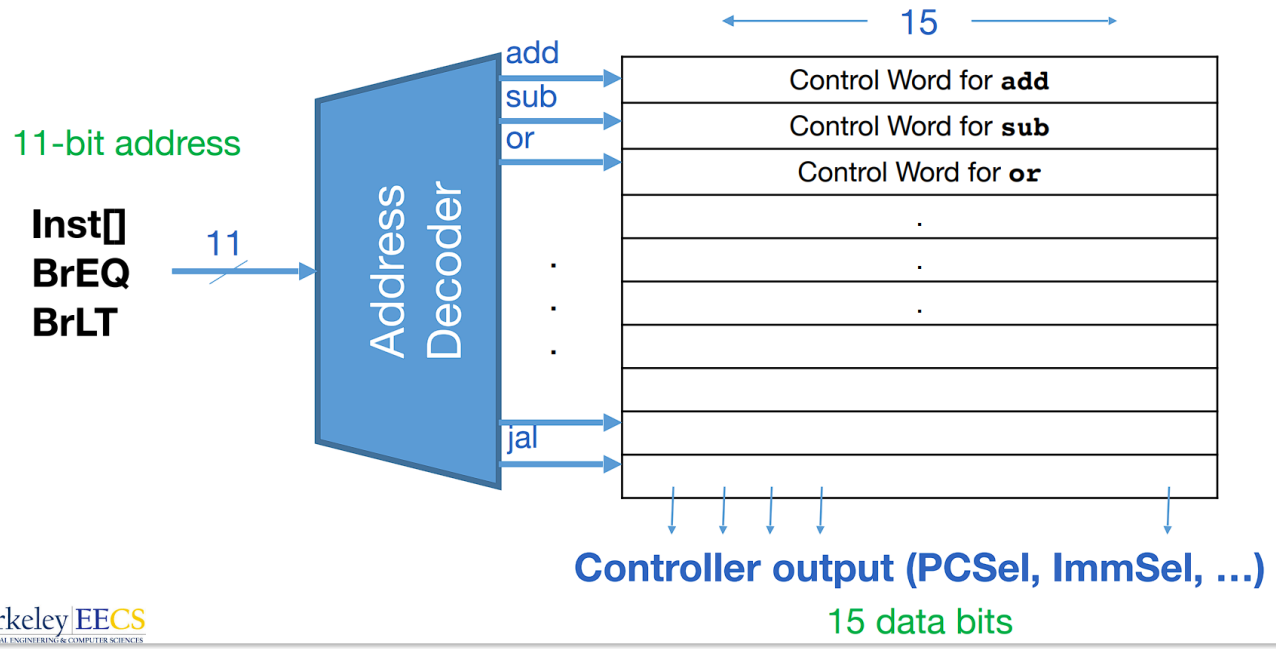
Control Block Design



Controller Realization Options

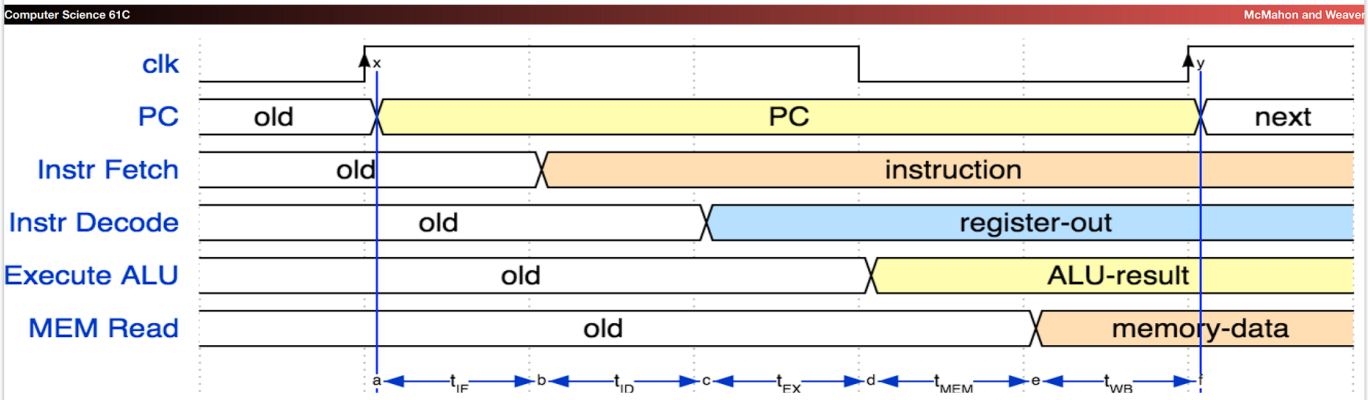
- ROM (Read-Only Memory)
  - Regular structure made from transistors
  - Can be easily reprogrammed during the design process to
    - fix errors
    - add instructions
  - Popular when designing control logic manually
- Combinatorial Logic
  - Today, chip designers often use logic synthesis tools to convert truth tables to networks of gates
  - Logic equation for each control signal (common sub-expressions shared among control signal equations)
  - Can exploit output “don’t cares” and input “for all values” to simplify circuit.

ROM (read only memory) Controller Implementation



Instruction Timing

Typical Approximate Worst-Case Instruction Timing



IF	ID	EX	MEM	WB	Total
I-MEM	Reg Read	ALU	D-MEM	Reg W	
200 ps	100 ps	200 ps	200 ps	100 ps	800 ps

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

18

- How can we keep **Data-Path** resources (ALU ,etc.) busy all the time.

Performance Measures

$$\frac{time}{program} = \frac{instructions}{program} \cdot \frac{cycles}{instruction} \cdot \frac{time}{cycle}$$

Instructions per Program

- Determined by
  - Task specification
  - Algorithm, e.g. O(N^2) vs O(N)
  - Programming language
  - Compiler
  - Instruction Set Architecture (ISA)

$$\frac{time}{program} = \boxed{\frac{instructions}{program}} \cdot \frac{cycles}{instruction} \cdot \frac{time}{cycle}$$

(Average) Clock cycles per Instruction

- Determined by
  - ISA (CISC versus RISC)
  - Processor implementation (or ***microarchitecture***)
    - E.g. for “our” single-cycle RISC-V design, CPI = 1
  - Pipelined processors, CPI >1 (next lecture)
  - Superscalar processors, CPI < 1 (next lecture)

$$\frac{time}{program} = \frac{instructions}{program} \cdot \boxed{\frac{cycles}{instruction}} \cdot \frac{time}{cycle}$$

Time per Cycle (1/Frequency)

$$\frac{time}{program} = \frac{instructions}{program} \cdot \frac{cycles}{instruction} \cdot \boxed{\frac{time}{cycle}}$$

- Determined by
  - Processor microarchitecture (determines critical path through logic gates)
  - Technology (e.g. 5nm versus 14nm)
  - Supply voltage (lower voltage reduces transistor speed, but improves energy efficiency)

- For some task (e.g. image compression) …

	Processor A	Processor B
# Instructions	1 Million	1.5 Million
Average CPI	2.5	1
Clock rate *f*	2.5 GHz	2 GHz
Execution time	1 ms	0.75 ms

Processor B is faster for this task, despite executing more instructions and having a lower clock rate!

$$\frac{energy}{program} = \frac{instructions}{program} \cdot \frac{energy}{instruction}$$

$$\frac{energy}{program} \propto \frac{instructions}{program} \cdot CV_{dd}^{\ 2}$$

"Capacitance" depends on technology, microarchitecture, circuit details

Supply voltage, e.g. 1V

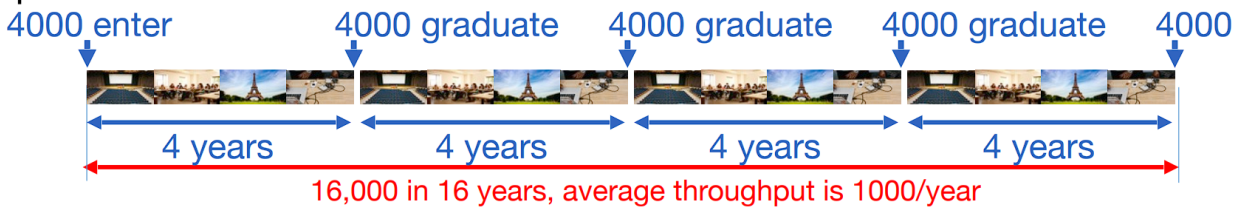
Want to reduce capacitance and voltage to reduce energy/task

Energy Tradeoff Example

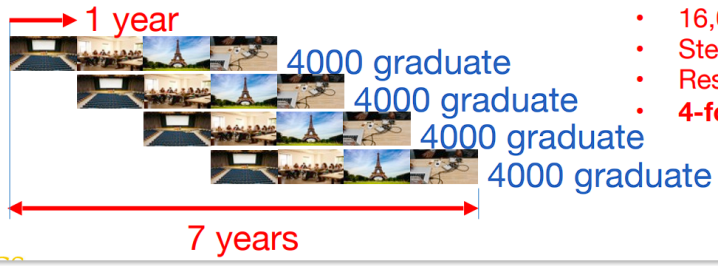
- For instance, "Next-generation" processor (Moore's law):
  - Capacitance, C: reduced by 15 %
  - Supply voltage, V_{sup}: reduced by 15 %
  - Energy consumption: $(.85C)(.85V)^2 = .63E$ => -39 % reduction
- Significantly improved energy efficiency thanks to
  - Moore's Law AND
  - Reduced supply voltage

Pipelining

- Option 1: ***serial***



- Option 2: ***pipelining***



- 16,000 in 7 years
- Steady state throughput is 4000/year
- Resources used efficiently
- **4-fold improvement over serial education**

- ***Latency***
  - Time from entering college to graduation
  - Serial 4 years
  - Pipelining 4 years
- ***Throughput***
  - Average number of students graduating each year
  - Serial 1000
  - Pipelining 4000
- *Pipelining*
  - Increases throughput (4x in this example)
  - But can ***never improve*** latency
    - sometimes worse (additional overhead)

Pipelining with RISC-V

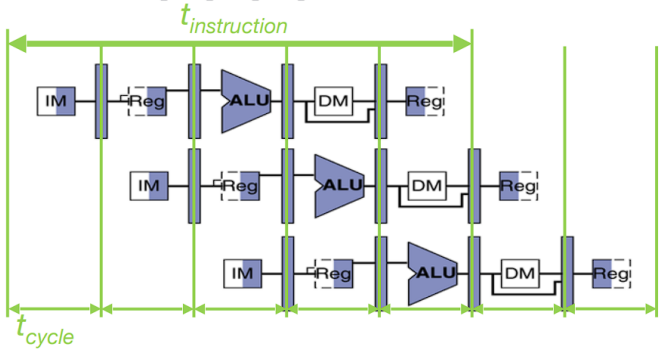
Phase	Pictogram	t_{step} Serial	t_{cycle} Pipelined
Instruction Fetch	IM	200 ps	200 ps
Reg Read	Reg	100 ps	200 ps
ALU	ALU	200 ps	200 ps
Memory	DM	200 ps	200 ps
Register Write	Reg	100 ps	200 ps
$t_{instruction}$	IM Reg ALU DM Reg	**800 ps**	**1000 ps**

instruction sequence

add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3



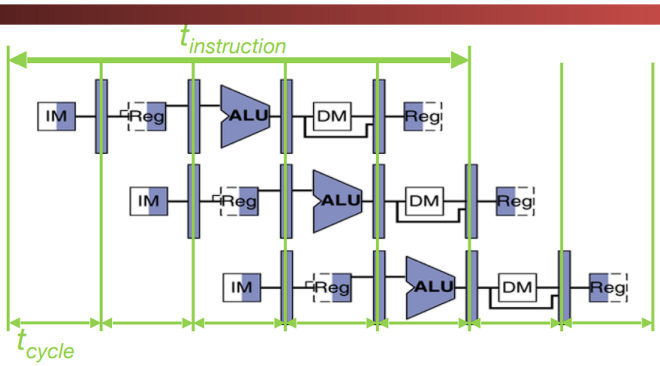
Pipelining with RISC-V

instruction sequence

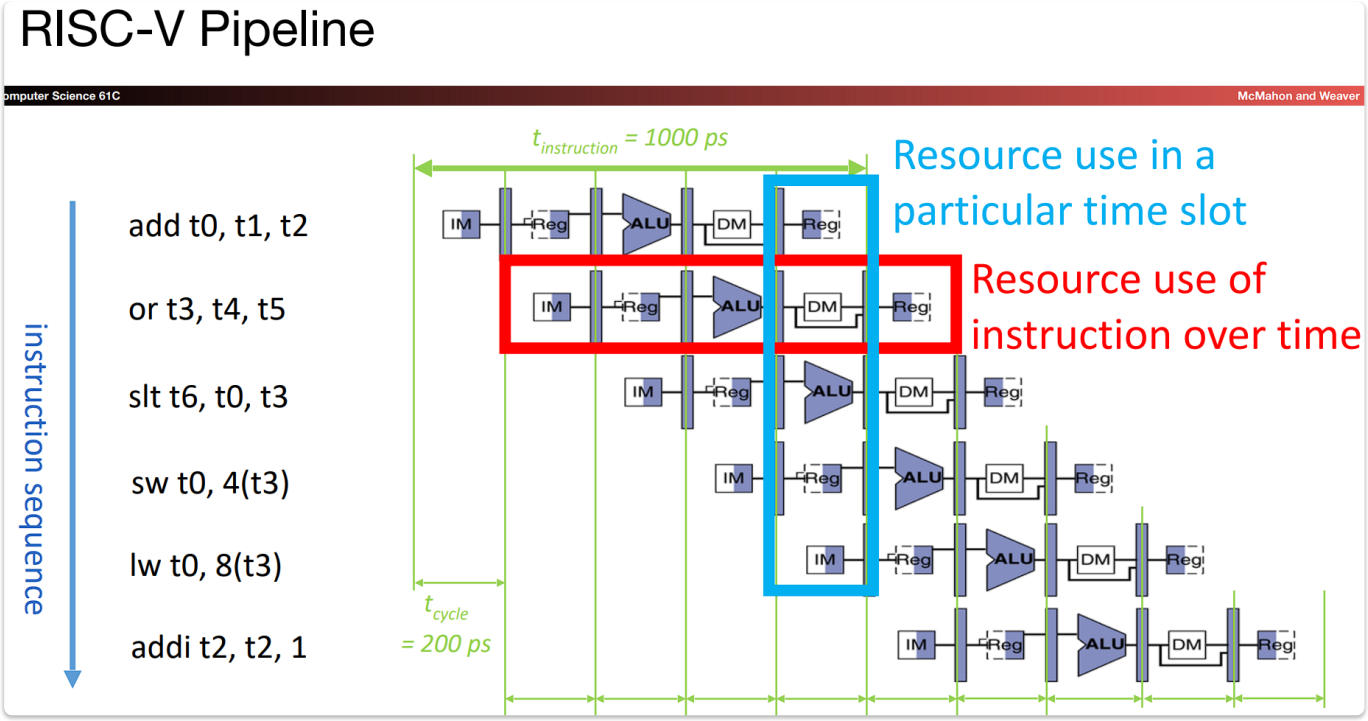
add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3

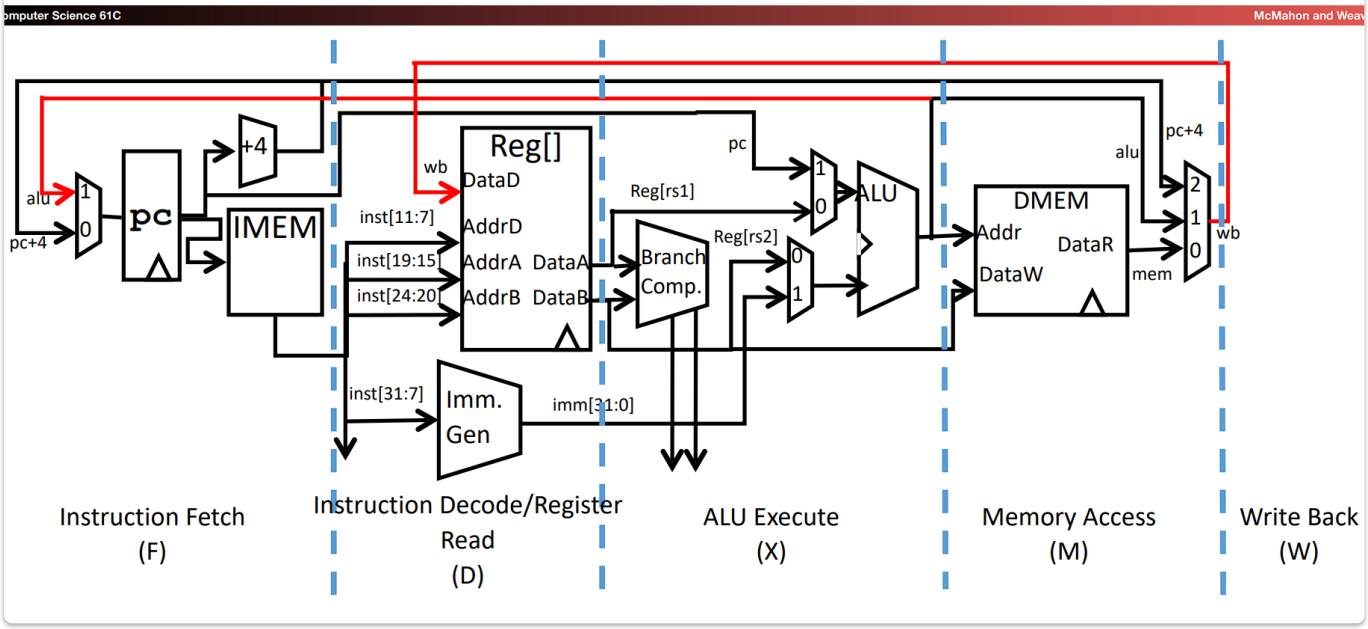


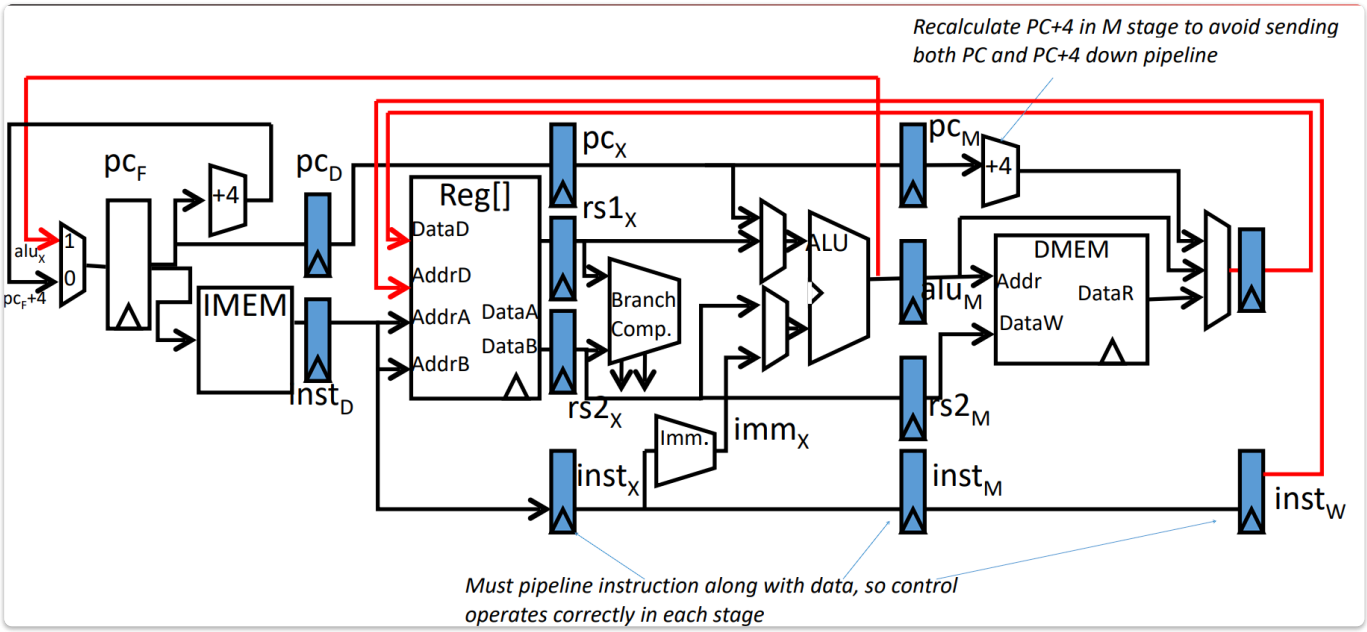
	Single Cycle	Pipelining
Timing	t_{step} = 100 … 200 ps	t_{cycle} = 200 ps
	(Register access only 100 ps)	All cycles same length
Instruction time, $t_{instruction}$	= t_{cycle} = 800 ps	1000 ps
Clock rate, f_s	1/800 ps = 1.25 GHz	1/200 ps = 5 GHz
Relative speed	1 x	4 x



Pipelining Datapath

Pipelining RISC-V RV32I Datapath





Pipelined Control

- Control signals derived from instruction
  - As in single-cycle implementation
  - Information is stored in pipeline registers for use by later stages