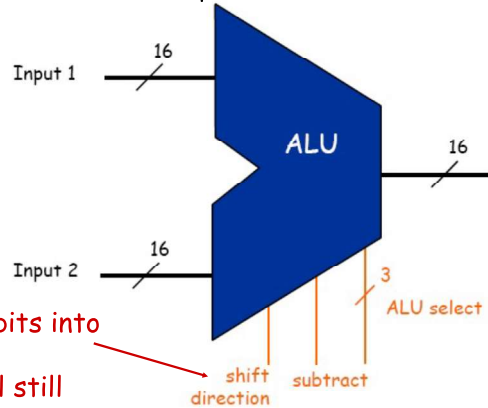


Recap: ALU

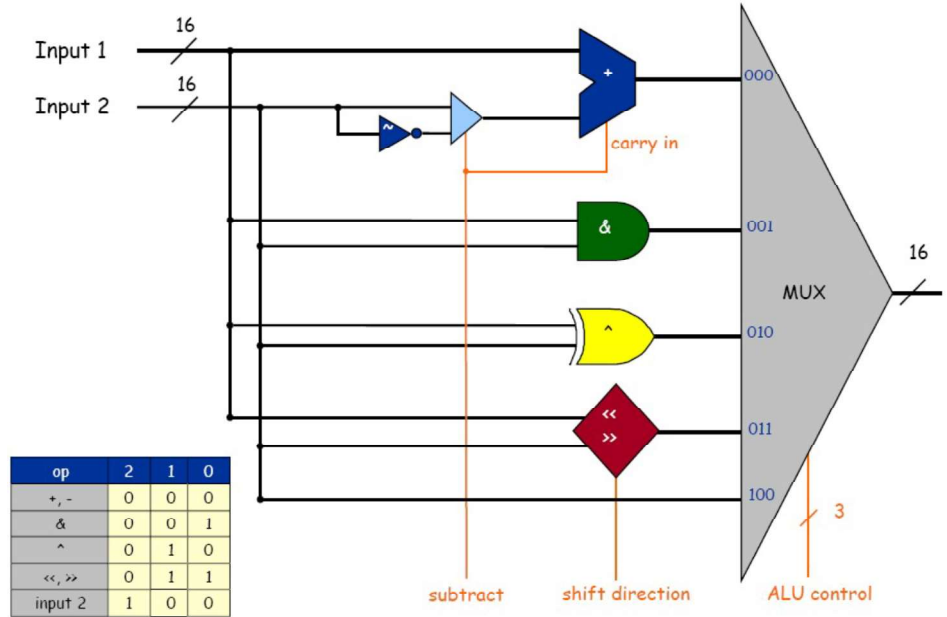
- Big combinational logic (16-bit bus)
- Add/subtract, and, xor, shift left/right, copy input 2
- A 3-bit control for 5 primary ALU operations
 - ALU performs operations in parallel
 - Control wires select which result ALU outputs

op	2	1	0
+, -	0	0	0
&	0	0	1
^	0	1	0
<<, >>	0	1	1
input 2	1	0	0



Can we combine these 5 bits into 3 bits for 7 operations?
 Yes, you can. But, you will still need 5 bits at the end.

Recap: ALU



op	2	1	0
+, -	0	0	0
&	0	0	1
^	0	1	0
<<, >>	0	1	1
input 2	1	0	0

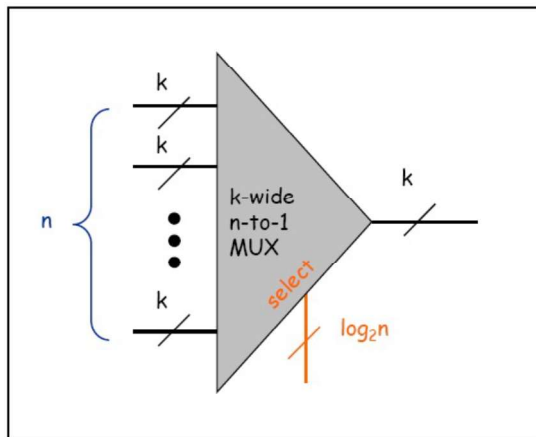
1

2

Recap: Multiplexer

Goal: select from one of n k -bit buses

- Implemented by layering k n -to-1 multiplexer

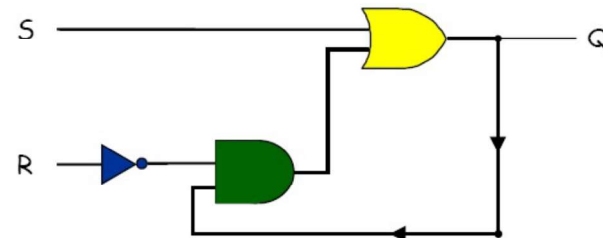


Interface

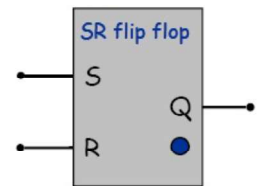
Recap: flip flop

SR Flip-Flop.

- $S = 1, R = 0$ (set) \Rightarrow Flips "bit" on.
- $S = 0, R = 1$ (reset) \Rightarrow Flips "bit" off.
- $S = R = 0$ \Rightarrow Status quo.
- $S = R = 1$ \Rightarrow Not allowed.



Implementation



Interface

3

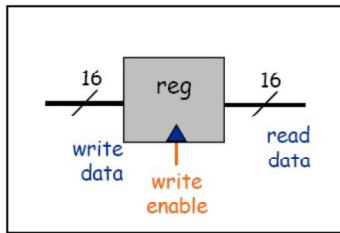
4

Stand-Alone Register

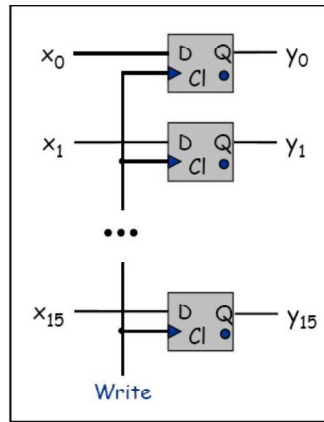
k-bit register.

- Stores k bits.
- Register contents always available on output.
- If write enable is asserted, k input bits get copied into register.

Ex: Program Counter, 16 TOY registers, 256 TOY memory locations.



16-bit Register Interface

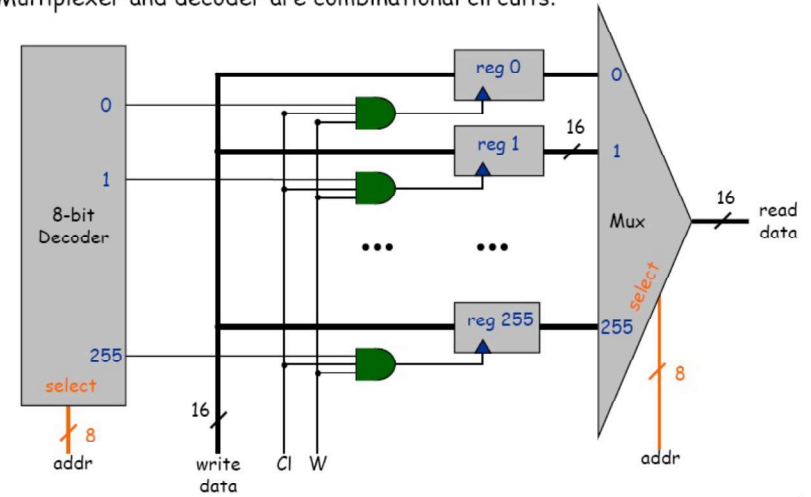


16-bit Register Implementation

Register file implementation

Implementation example: TOY main memory.

- Use 256 16-bit registers.
- Multiplexer and decoder are combinational circuits.

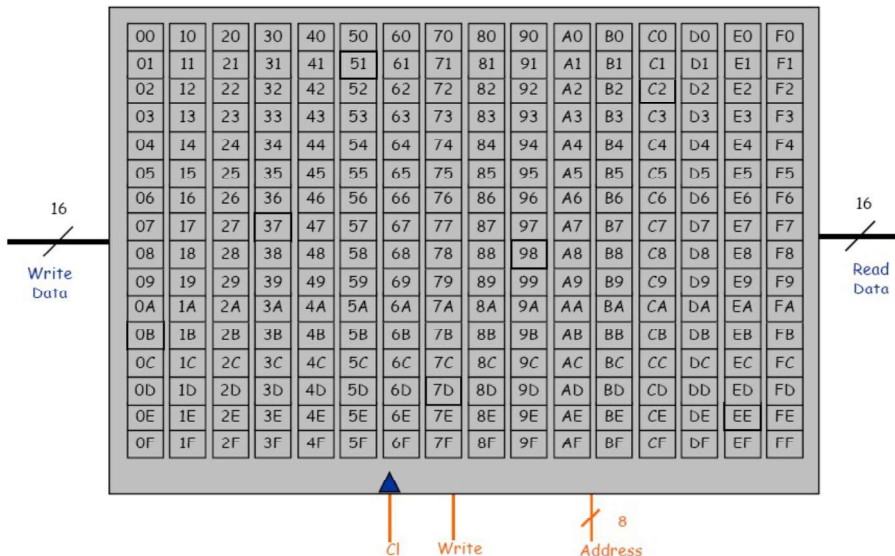


5

6

Recap: memory

TOY main memory: 256 x 16-bit register file.

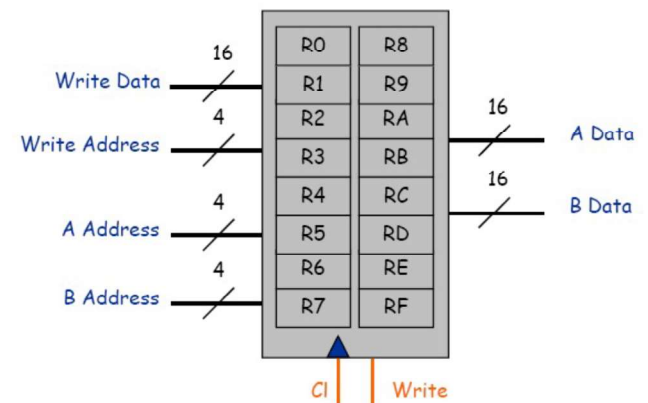


7

Recap: register file

TOY registers: fancy 16 x 16-bit register file.

- Want to be able to read two registers, and write to a third in the same instructions: $R1 \leftarrow R2 + R3$.
- 3 address inputs, 2 data outputs.
- Add extra bank of muxes for a second read port.

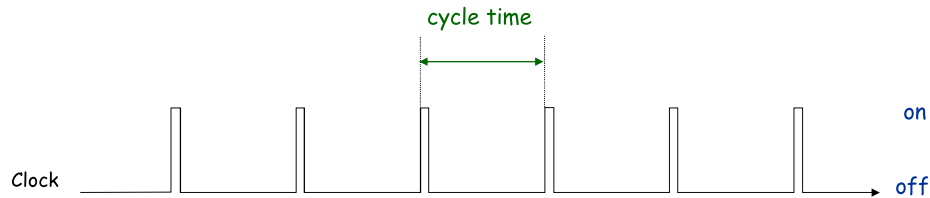


8

Clock

Clock.

- Fundamental abstraction: regular on-off pulse.
 - on: fetch phase
 - off: execute phase
- External analog device.
- Synchronizes operations of different circuit elements.
- Requirement: clock cycle longer than max switching time.



9

Introduction to Computer Science · Robert Sedgewick and Kevin Wayne · Copyright © 2005 · <http://www.cs.Princeton.EDU/IntroCS>

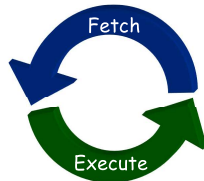
TOY Machine Architecture

The TOY Machine

Combinational circuits. ALU.
 Sequential circuits. Memory.
 Machine architecture. Wire components together to make computer.

TOY machine.

- 256 16-bit words of memory.
- 16 16-bit registers.
- 1 8-bit program counter.
- 16 instruction types.



11

Design a processor

How to build a processor

- Develop instruction set architecture (ISA)
 - 16-bit words, 16 TOY machine instructions
- Determine major components
 - ALU, memory, registers, program counter
- Determine datapath requirements
 - Flow of bits
- Analyze how to implement each instruction
 - Determine settings of control signals

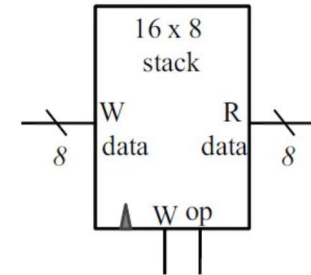
12

Practice: 4-bit counter



operation	op	semantics
reset	00	$C \leftarrow 0$
load	01	$C \leftarrow \text{in}$
inc	10	$C \leftarrow C+1$
dec	11	$C \leftarrow C-1$

Practice: stack



W	op	operation	semantics
0	0	read	the content of stack[top] can be read from "R data"
0	1	top	the content of top can be read from "R data"
1	0	push	top++; write the "W data" to stack[top];
1	1	pop	top--;

13

14

W	op	operation	semantics
0	0	read	the content of stack[top] can be read from "R data"
0	1	top	the content of top can be read from "R data"
1	0	push	top++; write the "W data" to stack[top];
1	1	pop	top--;

Design a processor

How to build a processor

- ➔ Develop instruction set architecture (ISA)
 - 16-bit words, 16 TOY machine instructions
- Determine major components
 - ALU, memory, registers, program counter
- Determine datapath requirements
 - Flow of bits
- Analyze how to implement each instruction
 - Determine settings of control signals

15

16

Build a TOY: Interface

Instruction set architecture (ISA).

- 16-bit words, 256 words of memory, 16 registers.
- Determine set of primitive instructions.
 - too narrow \Rightarrow cumbersome to program
 - too broad \Rightarrow cumbersome to build hardware
- 16 instructions.

Instructions	
0:	halt
1:	add
2:	subtract
3:	and
4:	xor
5:	shift left
6:	shift right
7:	load address

Instructions	
8:	load
9:	store
A:	load indirect
B:	store indirect
C:	branch zero
D:	branch positive
E:	jump register
F:	jump and link

TOY Reference Card

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Format 1	opcode				dest d			source s			source t					
Format 2	opcode				dest d			addr								

#	Operation	Fmt	Pseudocode
0:	halt	1	exit(0)
1:	add	1	$R[d] \leftarrow R[s] + R[t]$
2:	subtract	1	$R[d] \leftarrow R[s] - R[t]$
3:	and	1	$R[d] \leftarrow R[s] \& R[t]$
4:	xor	1	$R[d] \leftarrow R[s] \wedge R[t]$
5:	shift left	1	$R[d] \leftarrow R[s] \ll R[t]$
6:	shift right	1	$R[d] \leftarrow R[s] \gg R[t]$
7:	load addr	2	$R[d] \leftarrow \text{addr}$
8:	load	2	$R[d] \leftarrow \text{mem}[\text{addr}]$
9:	store	2	$\text{mem}[\text{addr}] \leftarrow R[d]$
A:	load indirect	1	$R[d] \leftarrow \text{mem}[R[t]]$
B:	store indirect	1	$\text{mem}[R[t]] \leftarrow R[d]$
C:	branch zero	2	if $(R[d] == 0)$ pc \leftarrow addr
D:	branch positive	2	if $(R[d] > 0)$ pc \leftarrow addr
E:	jump register	1	pc $\leftarrow R[t]$
F:	jump and link	2	$R[d] \leftarrow \text{pc}; \text{pc} \leftarrow \text{addr}$

Register 0 always 0.
Loads from mem[FF]
from stdin.
Stores to mem[FF] to
stdout.

17

18

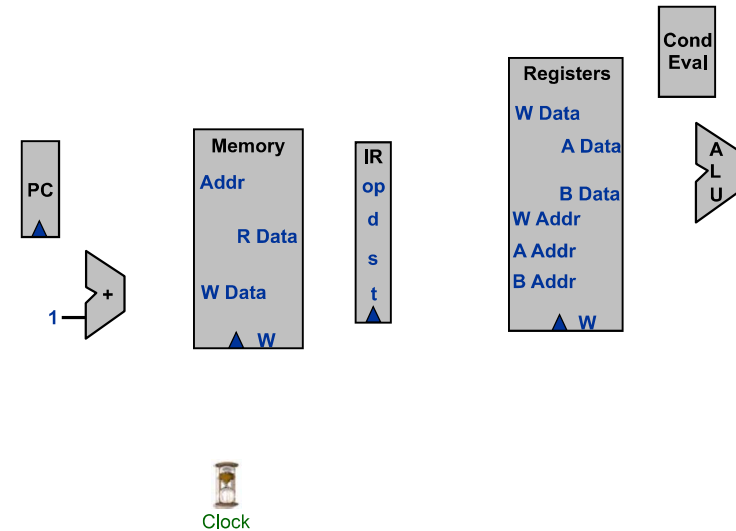
Design a processor

How to build a processor

- Develop instruction set architecture (ISA)
 - 16-bit words, 16 TOY machine instructions
- Determine major components
 - ALU, memory, registers, program counter
- Determine datapath requirements
 - Flow of bits
- Analyze how to implement each instruction
 - Determine settings of control signals

Components

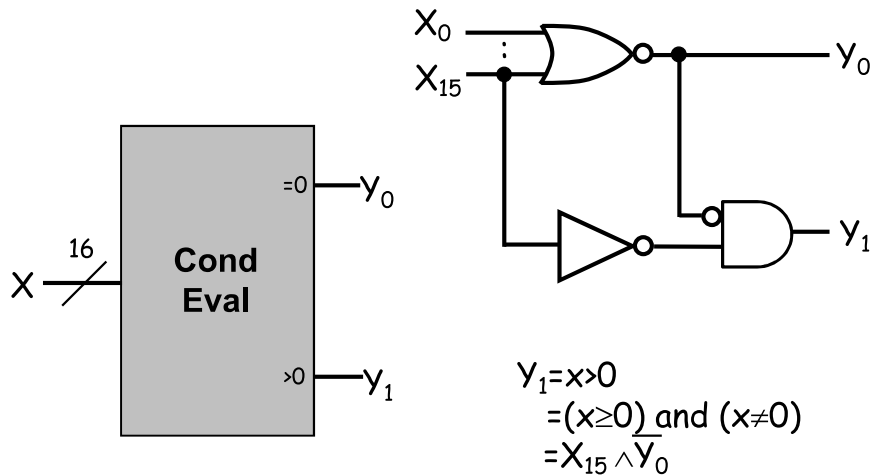
- Determine major components
 - ALU, memory, registers, program counter
- Determine datapath requirements
 - Flow of bits
- Analyze how to implement each instruction
 - Determine settings of control signals



19

20

Cond. Eval.



$$\begin{aligned}
 Y_1 &= x > 0 \\
 &= (x \geq 0) \text{ and } (x \neq 0) \\
 &= X_{15} \wedge \overline{Y_0}
 \end{aligned}$$

Design a processor

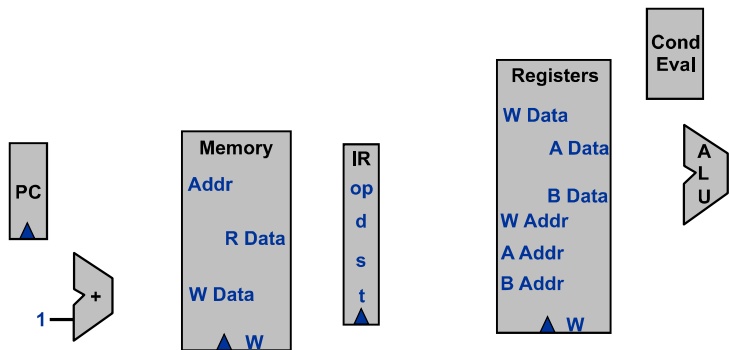
How to build a processor

- Develop instruction set architecture (ISA)
 - 16-bit words, 16 TOY machine instructions
- Determine major components
 - ALU, memory, registers, program counter
- ➔ Determine datapath requirements
 - Flow of bits
- Analyze how to implement each instruction
 - Determine settings of control signals

21

22

Datapath

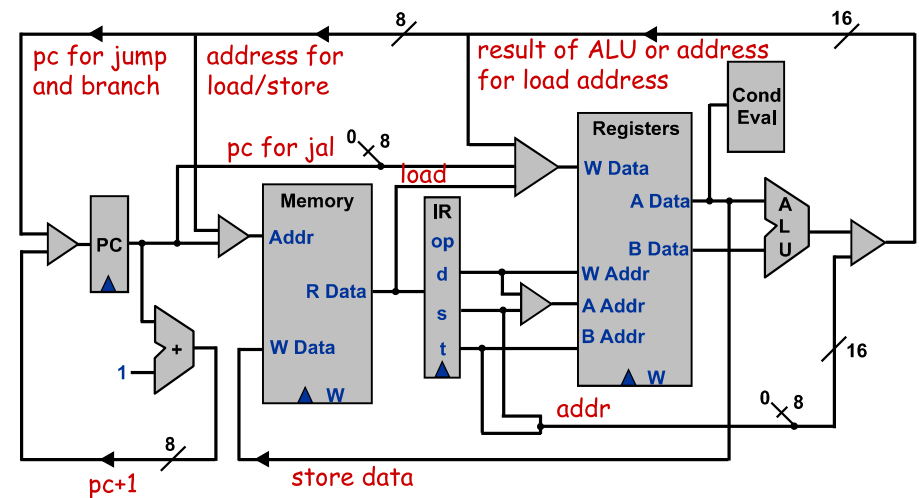


instruction fetch: $IR \leftarrow mem[PC]; pc \leftarrow pc + 1$

- | | | |
|------------------------------------|-------------------------------|--|
| 1-6 $R[d] \leftarrow R[s]$ ALU | 7 $R[d] \leftarrow addr$ | 8 $R[d] \leftarrow mem[addr]$ |
| 9 $mem[addr] \leftarrow R[d]$ | A $R[d] \leftarrow mem[R[t]]$ | B $mem[R[t]] \leftarrow R[d]$ |
| CD if (R[d]?) $pc \leftarrow addr$ | E $pc \leftarrow R[t]$ | F $R[d] \leftarrow pc; pc \leftarrow addr$ |

23

Datapath



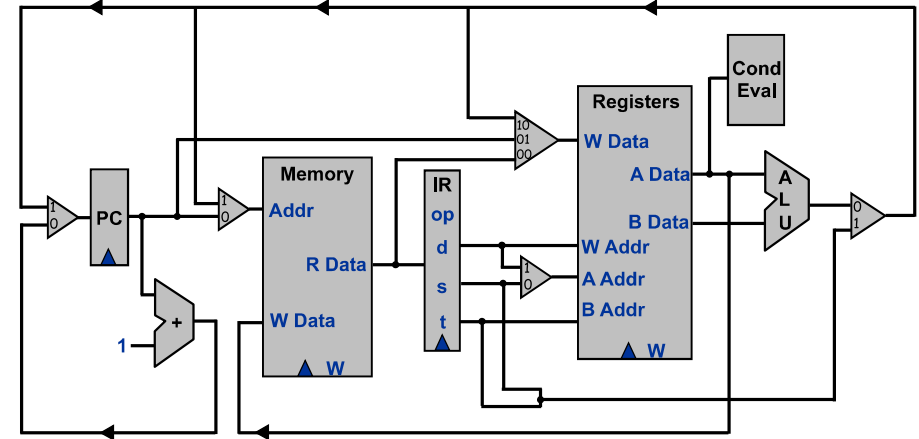
24

Design a processor

How to build a processor

- Develop instruction set architecture (ISA)
 - 16-bit words, 16 TOY machine instructions
- Determine major components
 - ALU, memory, registers, program counter
- Determine datapath requirements
 - Flow of bits
- ➔ Analyze how to implement each instruction
 - Determine settings of control signals

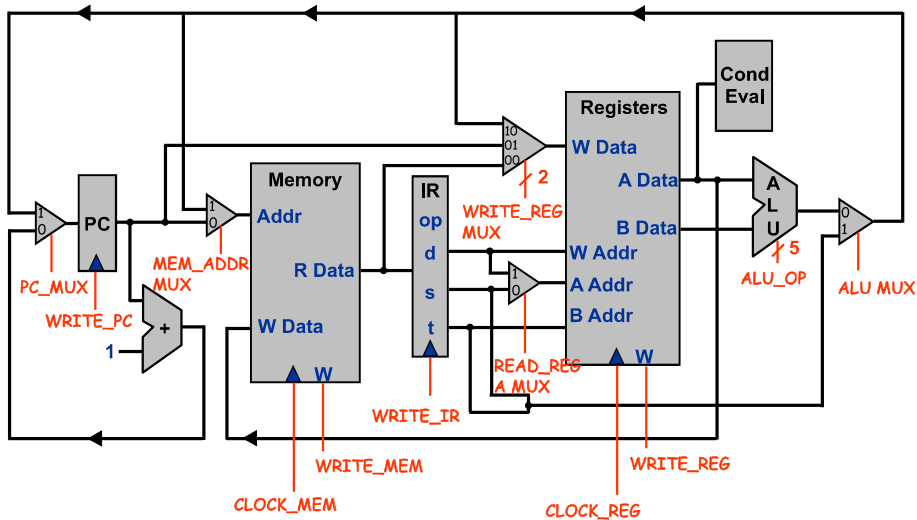
Datapath



25

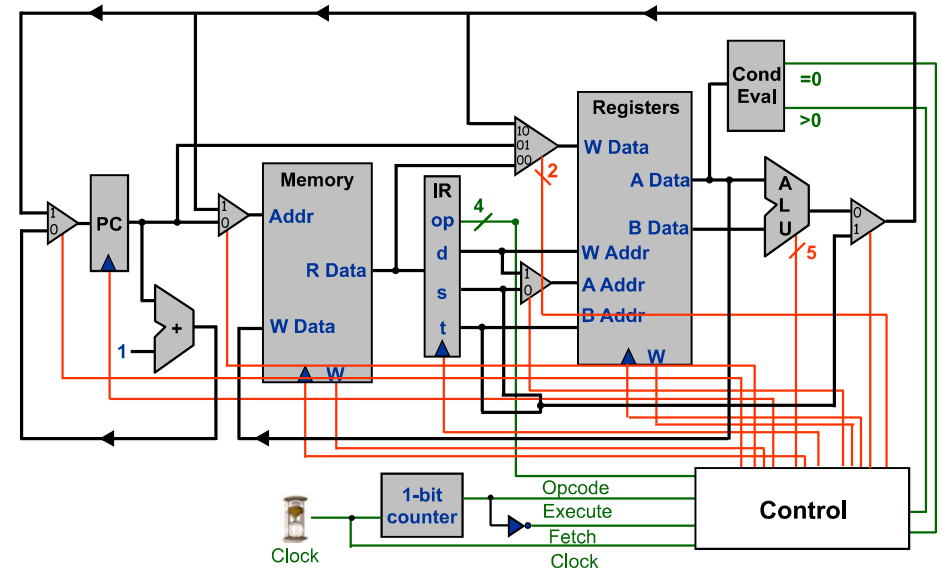
26

Control



A total of 17 control signals

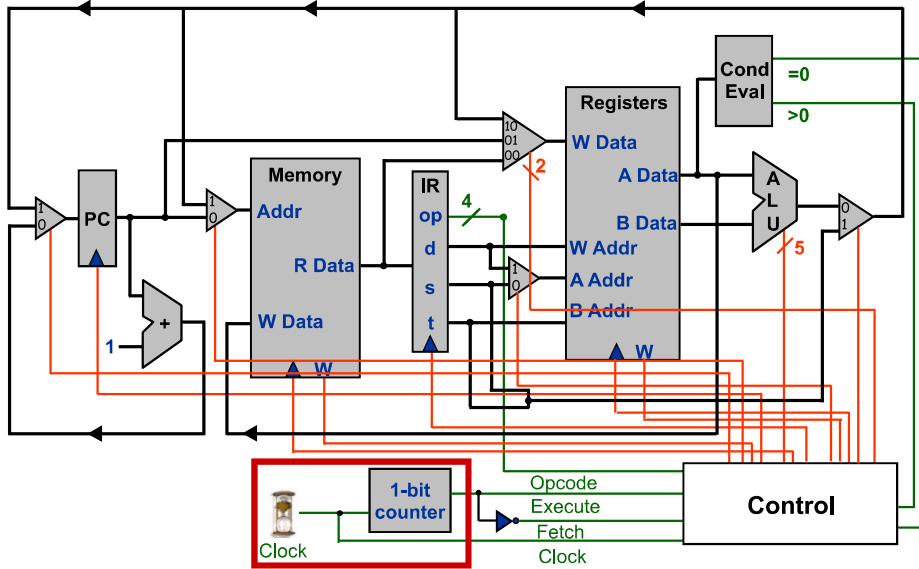
TOY architecture



27

28

Clock

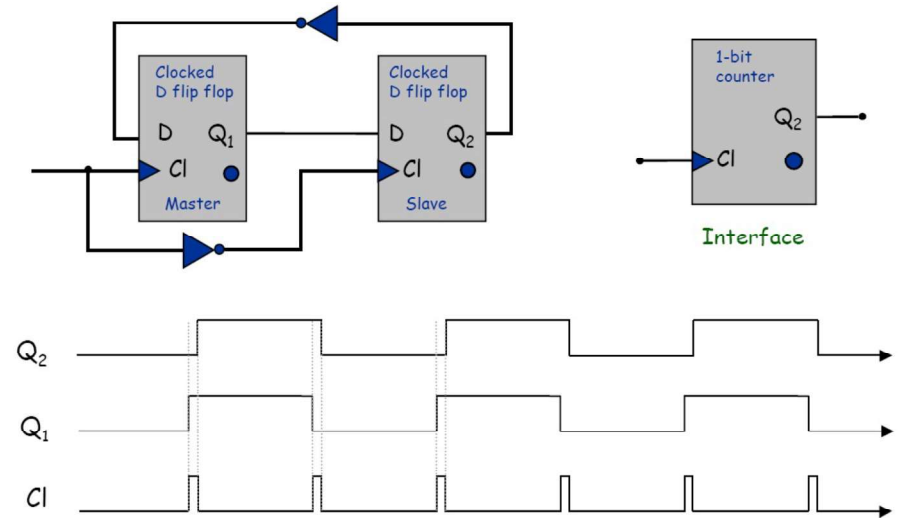


29

1-bit counter

1-bit counter

- Circuit that oscillates between 1 and 0.

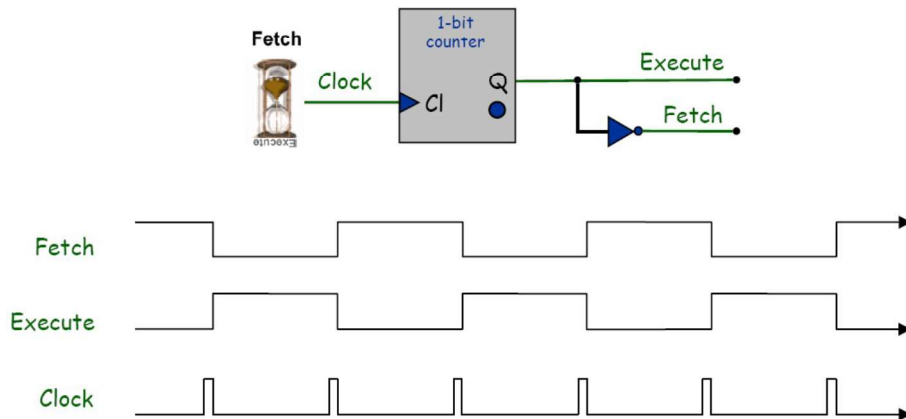


30

Clock

Two cycle design (fetch and execute)

- Use 1-bit counter to distinguish between 2 cycles.
- Use two cycles since fetch and execute phases each access memory and alter program counter.

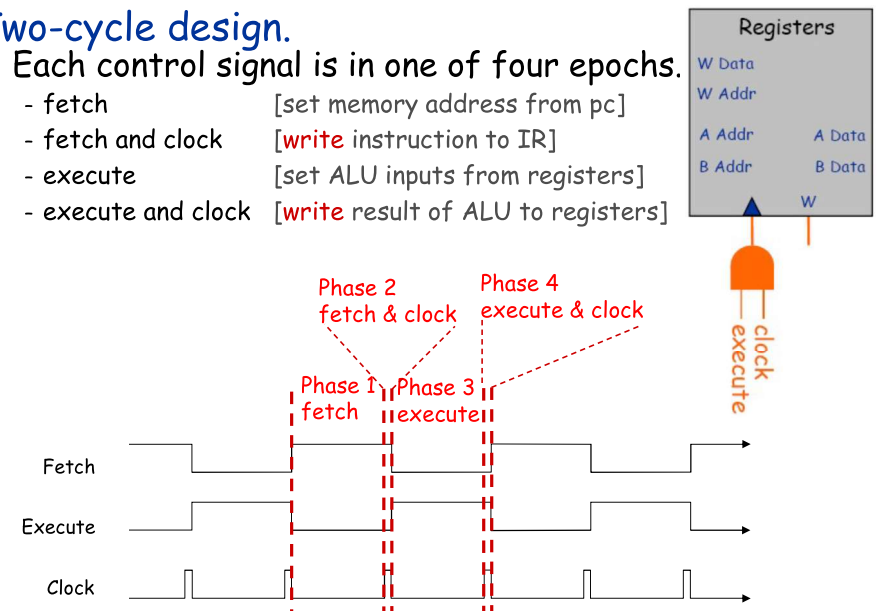


31

Clocking Methodology

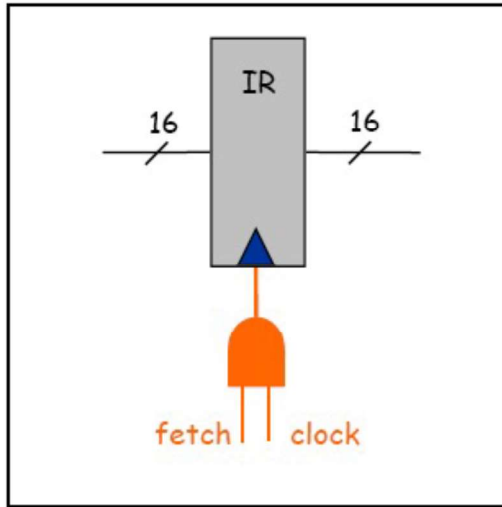
Two-cycle design.

- Each control signal is in one of four epochs.
 - fetch [set memory address from pc]
 - fetch and clock [write instruction to IR]
 - execute [set ALU inputs from registers]
 - execute and clock [write result of ALU to registers]



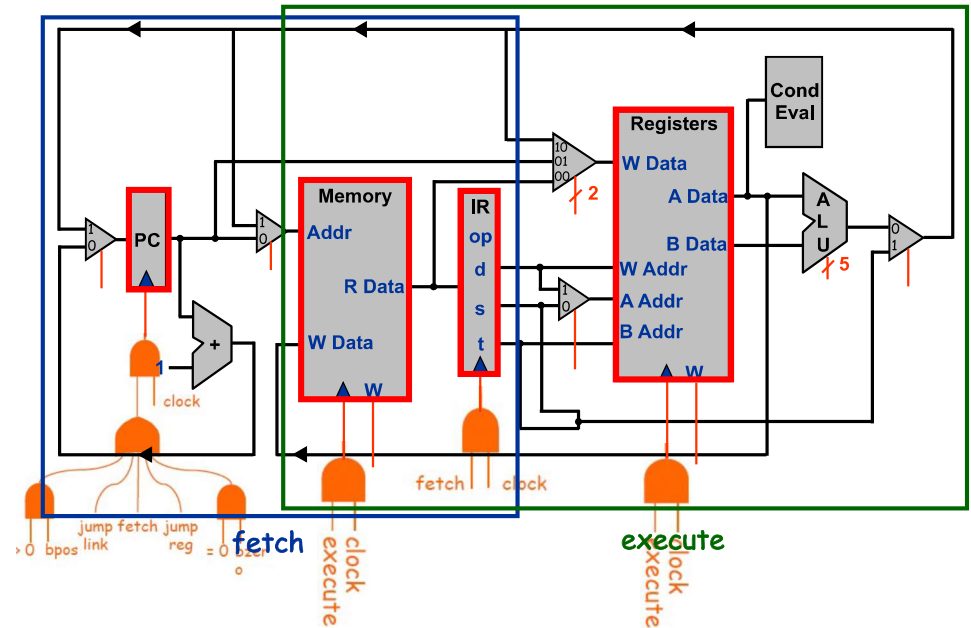
32

Instruction register



Instruction Register

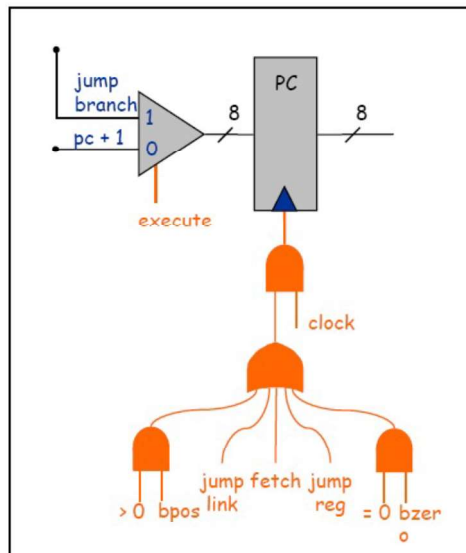
Clocking Methodology



33

34

Program counter



Program Counter

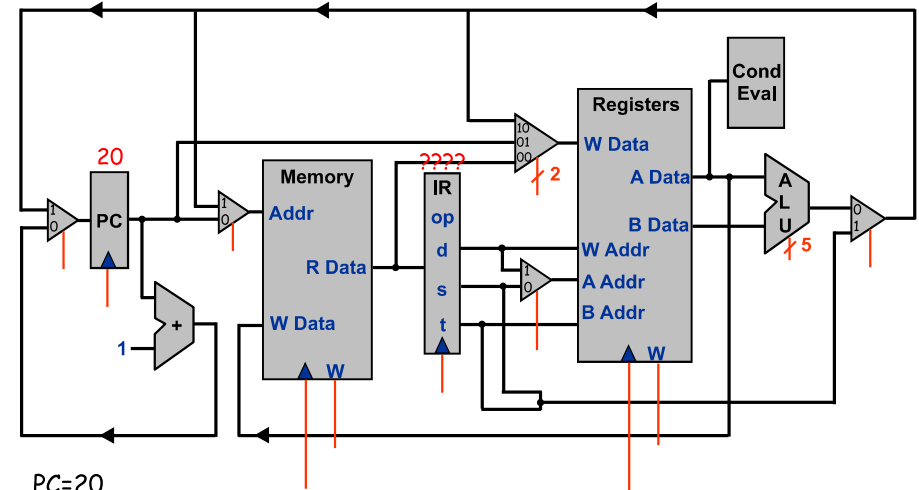
Read program counter when

- Fetch
- Execute for jal

Write program counter when

- Fetch and clock
- Execute and clock depending on conditions

Example: ADD

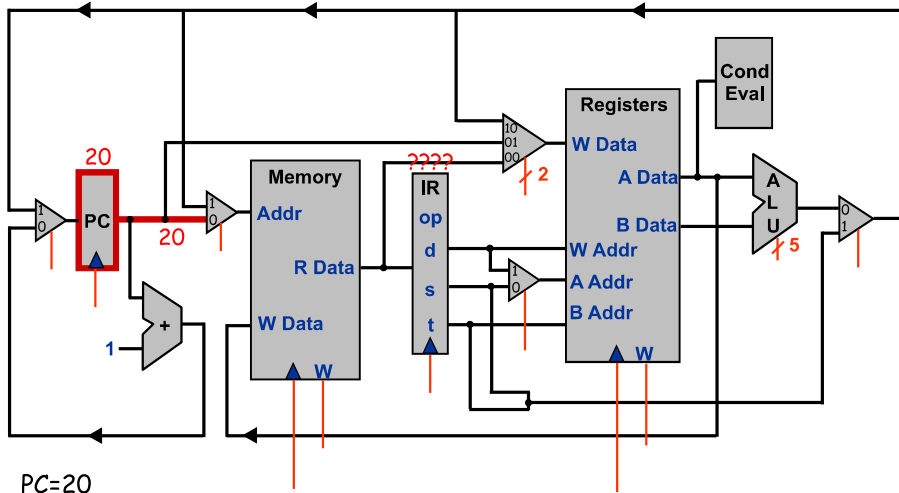


PC=20
Mem[20]=1234
R[3]=0028 R[4]=0064

35

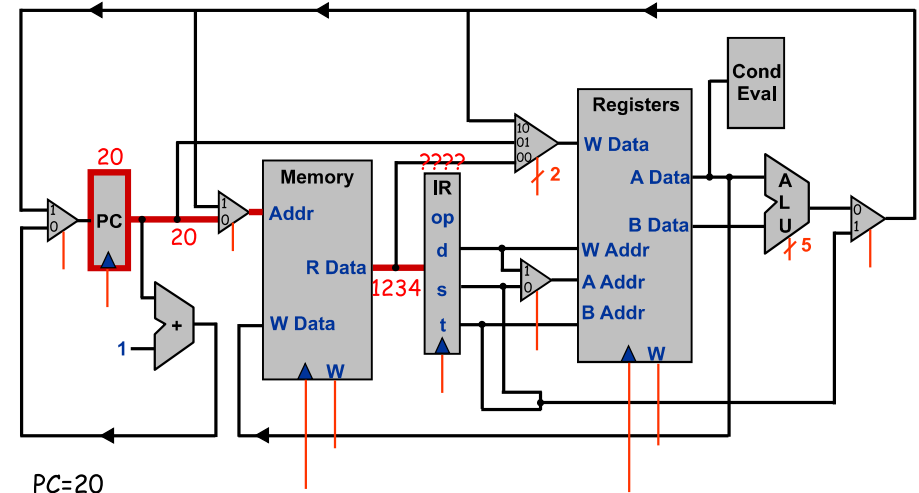
36

Example: ADD (fetch)



PC=20
 Mem[20]=1234
 R[3]=0028 R[4]=0064

Example: ADD (fetch)

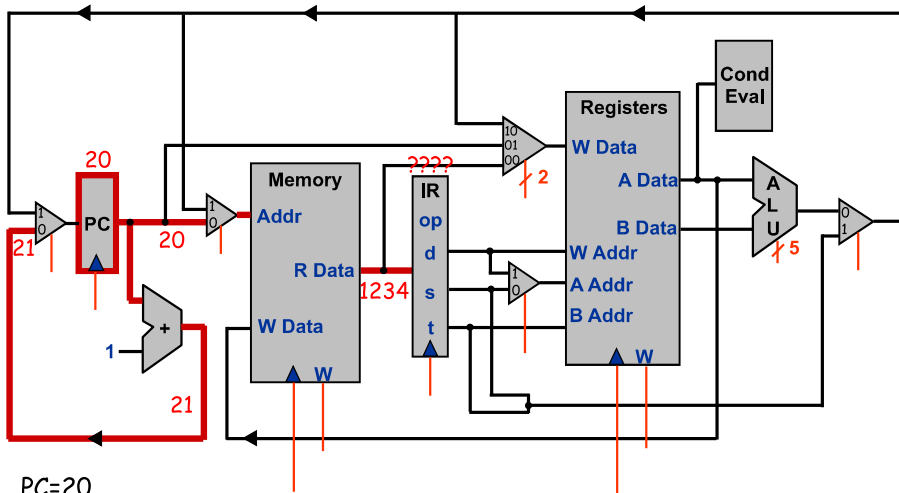


PC=20
 Mem[20]=1234
 R[3]=0028 R[4]=0064

37

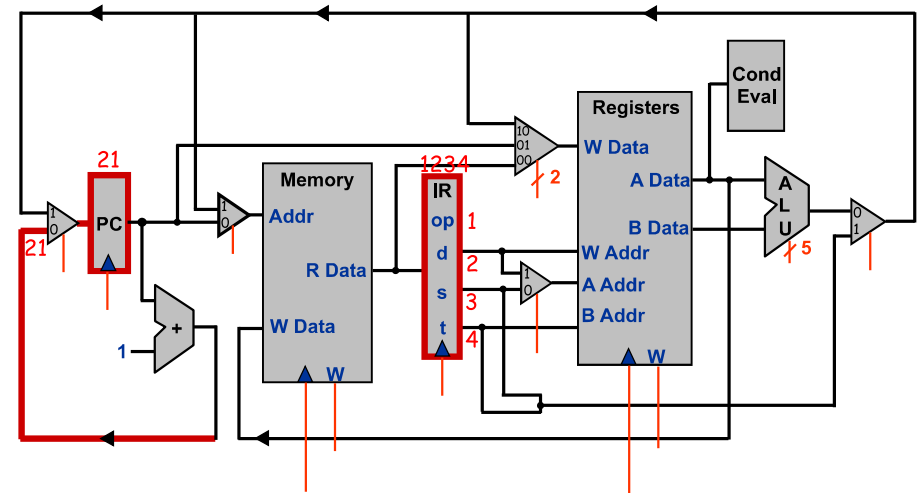
38

Example: ADD (fetch)



PC=20
 Mem[20]=1234
 R[3]=0028 R[4]=0064

Example: ADD (fetch and clock)

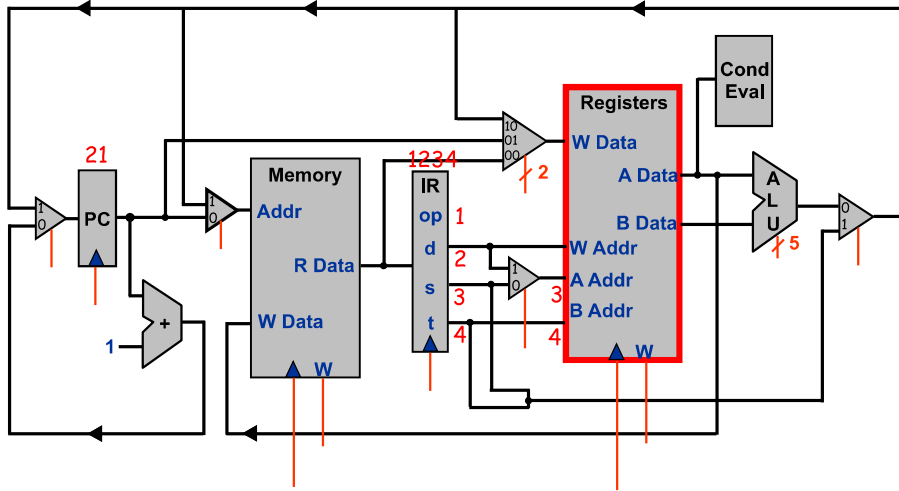


Mem[20]=1234
 R[3]=0028 R[4]=0064
 PC=21
 IR=1234

39

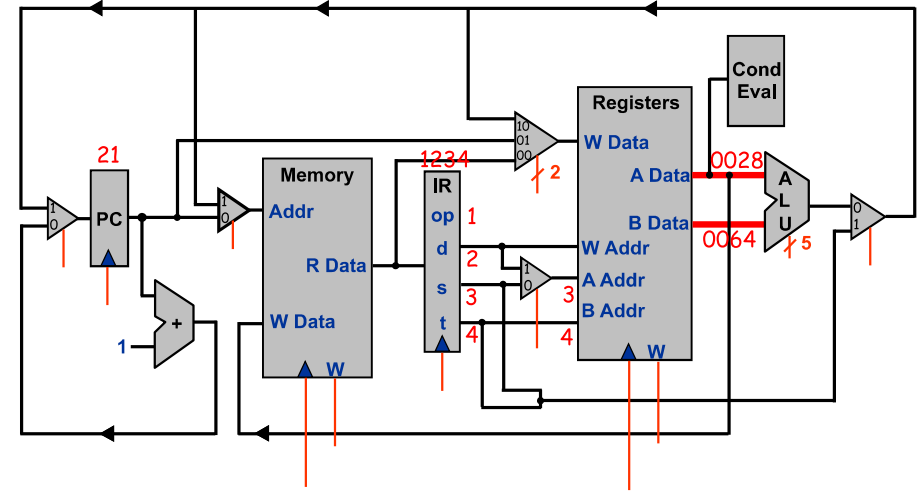
40

Example: ADD (execute)



Mem[20]=1234 PC=21
R[3]=0028 R[4]=0064 IR=1234

Example: ADD (execute)

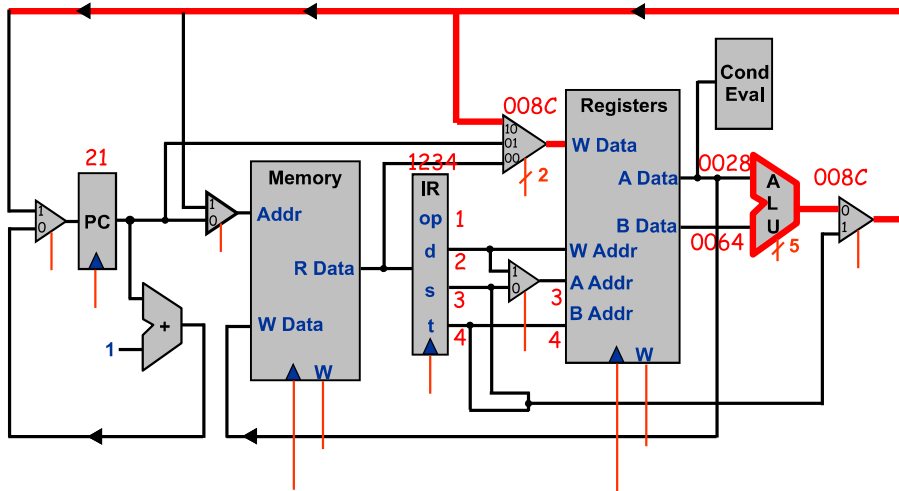


Mem[20]=1234 PC=21
R[3]=0028 R[4]=0064 IR=1234

41

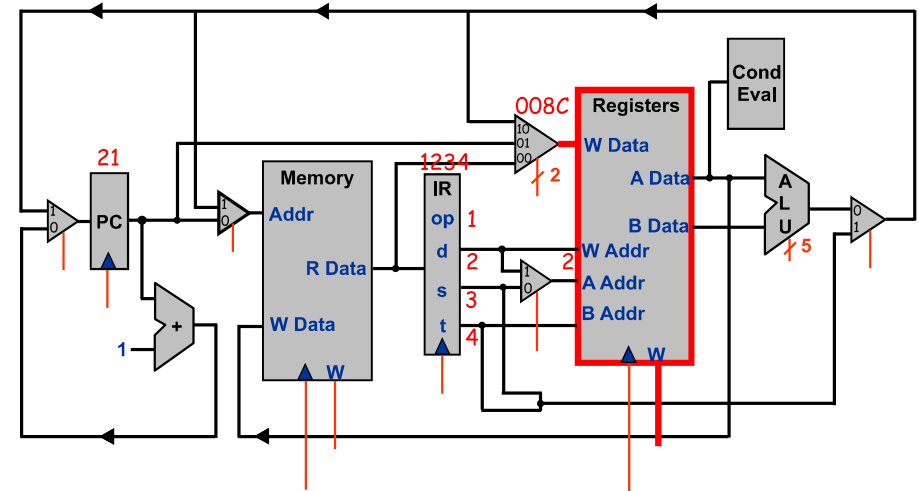
42

Example: ADD (execute)



Mem[20]=1234 PC=21
R[3]=0028 R[4]=0064 IR=1234

Example: ADD (execute and clock)

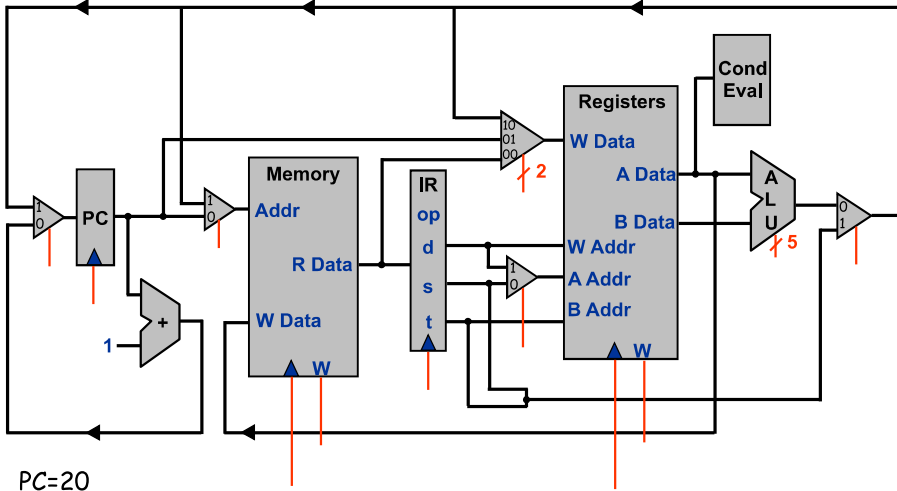


Mem[20]=1234 PC=21
R[3]=0028 R[4]=0064 R[2]=008C

43

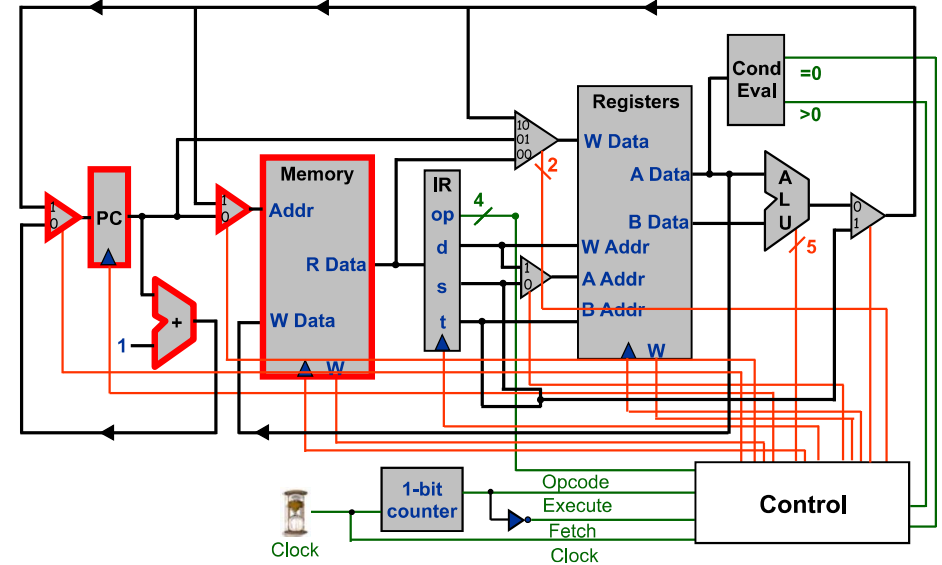
44

Example: Jump and link



PC=20
 Mem[20]=FF30
 R[3]=0028 R[4]=0064

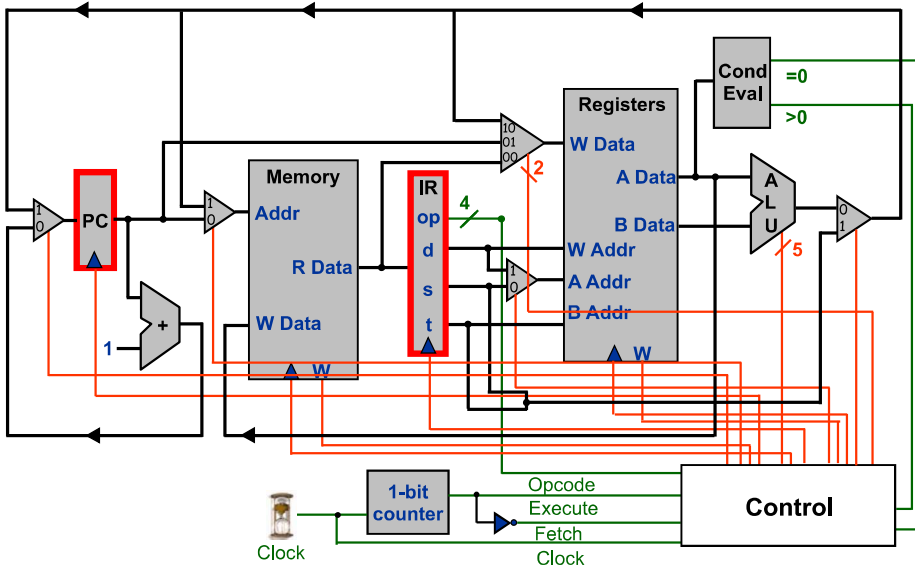
Fetch



45

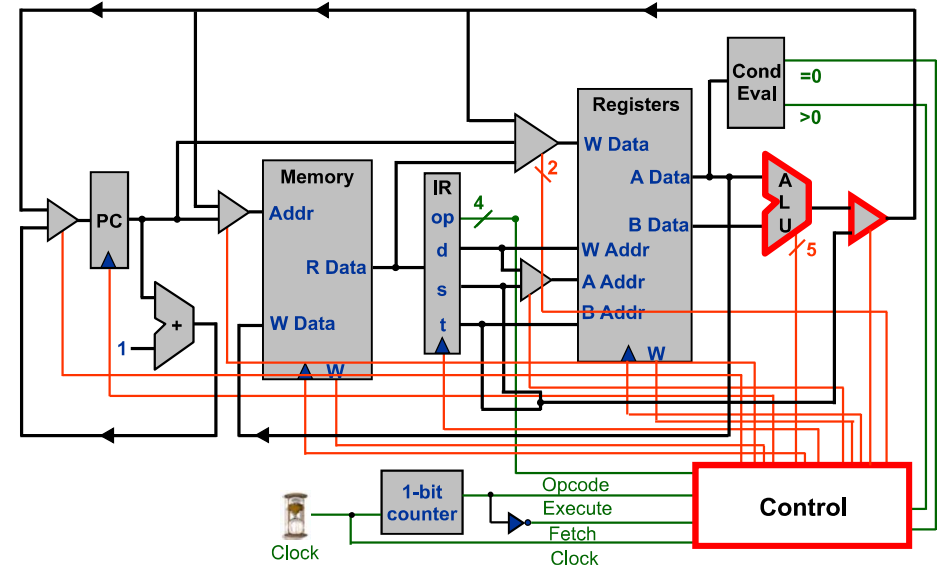
46

Fetch and clock



47

Execute

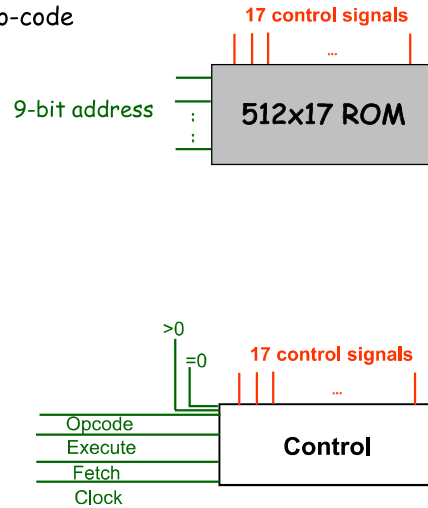


48

Control

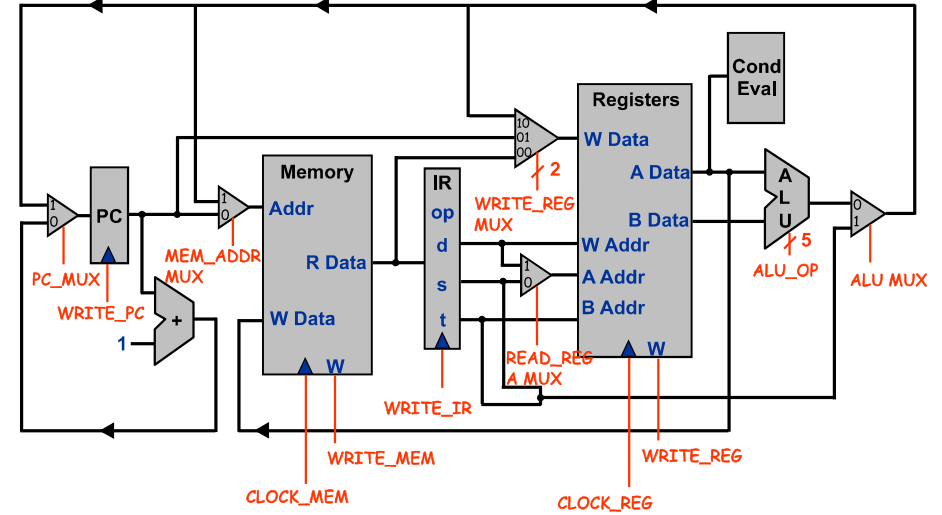
Two approaches to implement control

- **Micro-programming**
 - Use a memory (ROM) for micro-code
 - More flexible
 - Easier to program
- **Hard-wired**
 - Use logic gates and wire
 - More efficient



49

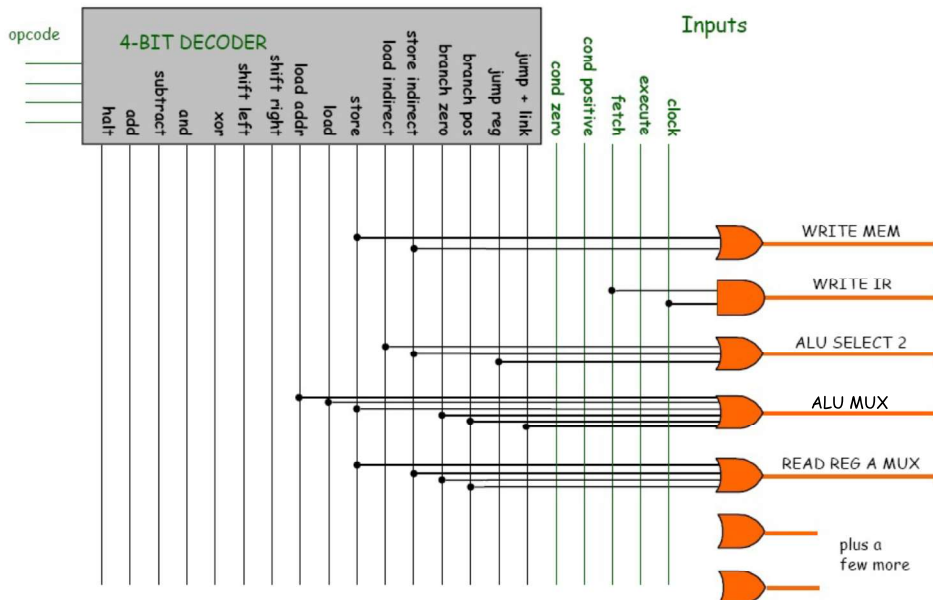
Control



A total of 17 control signals

50

Control



51

TOY Reference Card

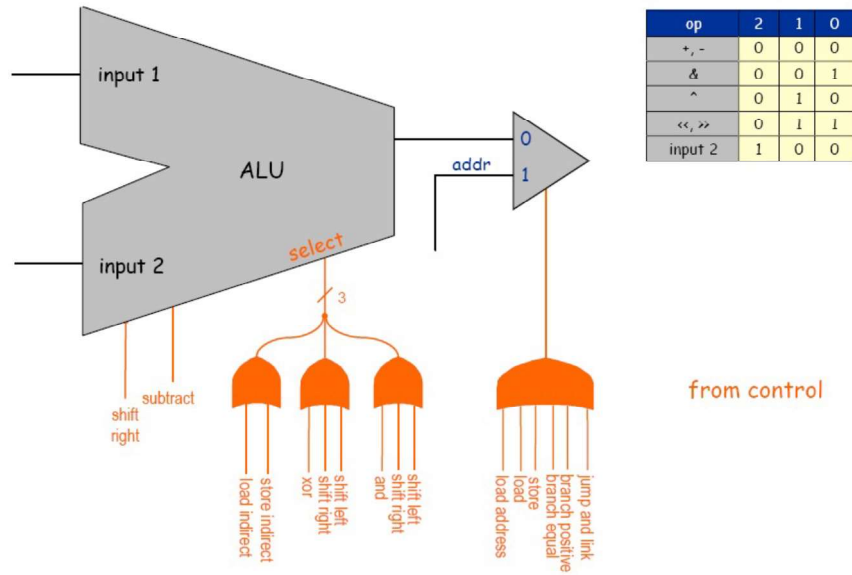
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Format 1	opcode				dest d			source s				source t				
Format 2	opcode				dest d			addr								

#	Operation	Fmt	Pseudocode
0:	halt	1	exit(0)
1:	add	1	$R[d] \leftarrow R[s] + R[t]$
2:	subtract	1	$R[d] \leftarrow R[s] - R[t]$
3:	and	1	$R[d] \leftarrow R[s] \& R[t]$
4:	xor	1	$R[d] \leftarrow R[s] \wedge R[t]$
5:	shift left	1	$R[d] \leftarrow R[s] \ll R[t]$
6:	shift right	1	$R[d] \leftarrow R[s] \gg R[t]$
7:	load addr	2	$R[d] \leftarrow \text{addr}$
8:	load	2	$R[d] \leftarrow \text{mem}[\text{addr}]$
9:	store	2	$\text{mem}[\text{addr}] \leftarrow R[d]$
A:	load indirect	1	$R[d] \leftarrow \text{mem}[R[t]]$
B:	store indirect	1	$\text{mem}[R[t]] \leftarrow R[d]$
C:	branch zero	2	if $(R[d] == 0)$ $\text{pc} \leftarrow \text{addr}$
D:	branch positive	2	if $(R[d] > 0)$ $\text{pc} \leftarrow \text{addr}$
E:	jump register	1	$\text{pc} \leftarrow R[t]$
F:	jump and link	2	$R[d] \leftarrow \text{pc}; \text{pc} \leftarrow \text{addr}$

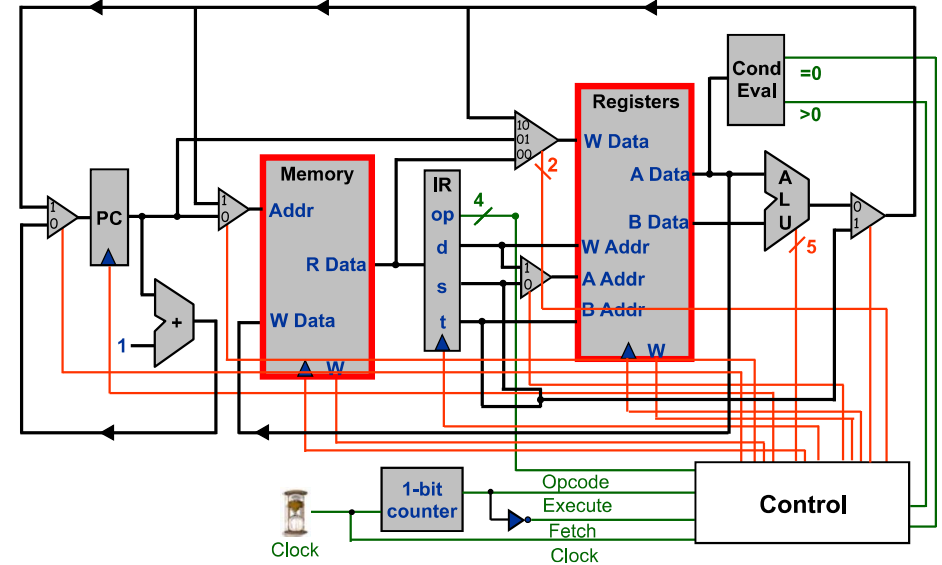
Register 0 always 0.
Loads from mem[FF]
from stdin.
Stores to mem[FF] to
stdout.

52

ALU control



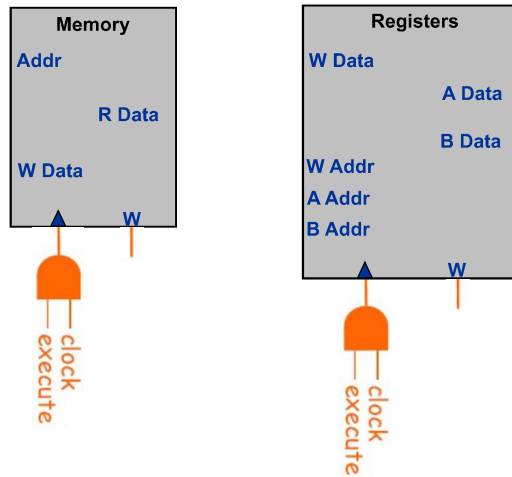
Execute and clock (write-back)



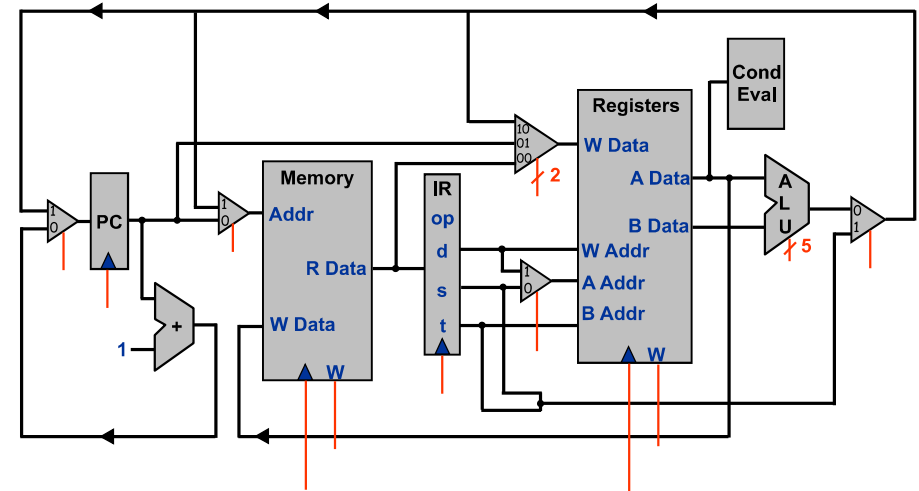
53

54

Writing registers and memory



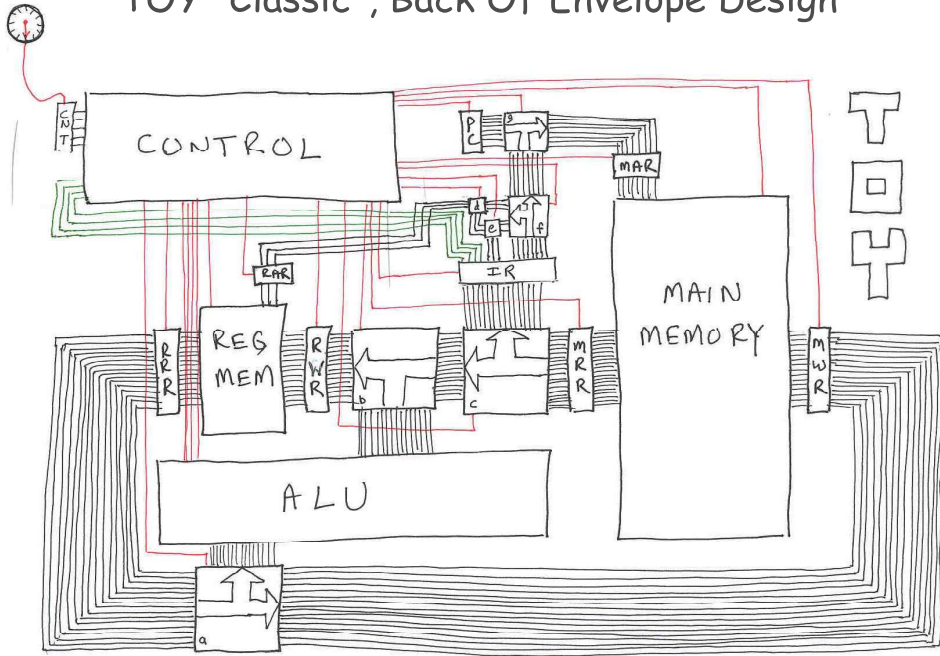
More examples



55

56

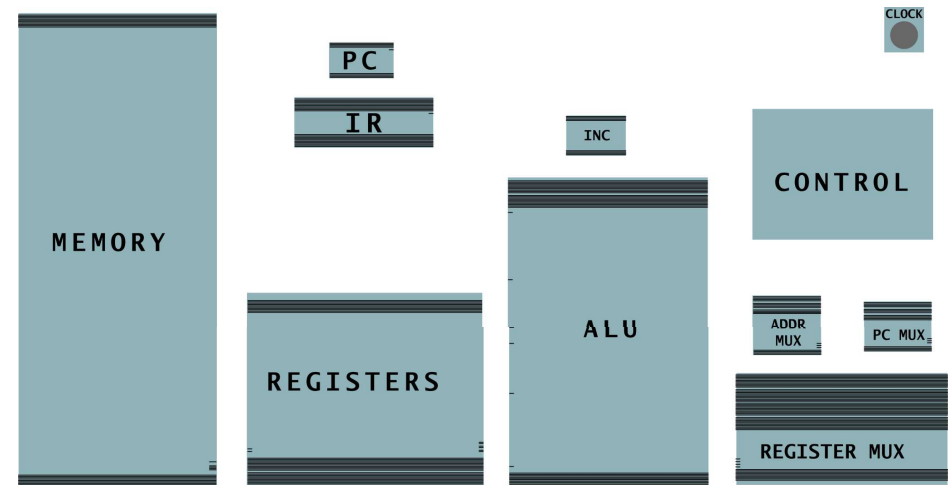
TOY "Classic", Back Of Envelope Design



57

Build a TOY-Lite: Devices

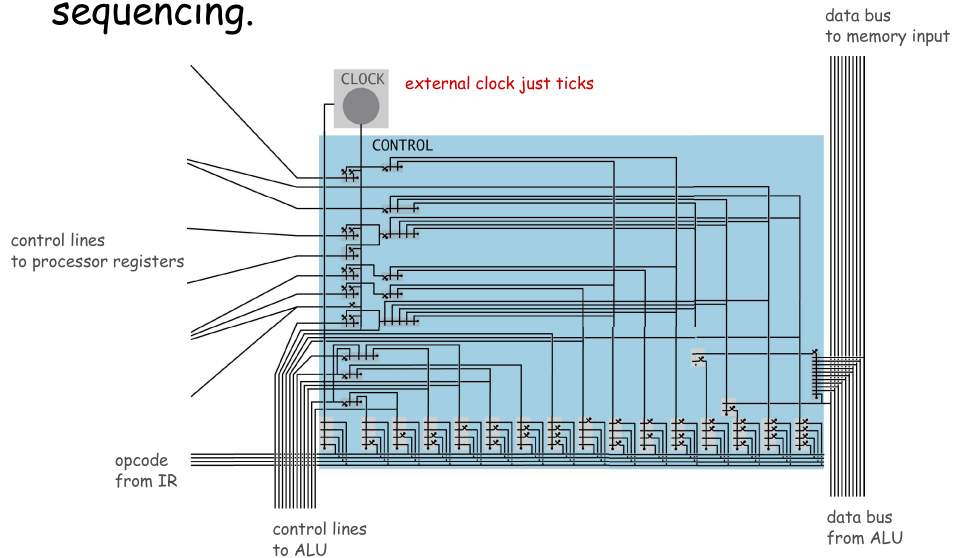
10-bit word,
4-word register
16-word memory



58

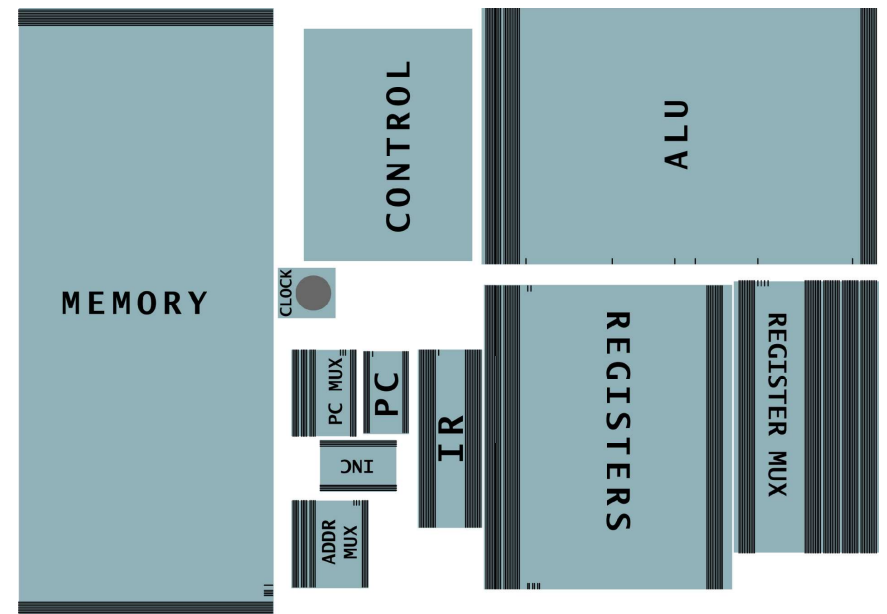
Control

Control. Circuit that determines control line sequencing.



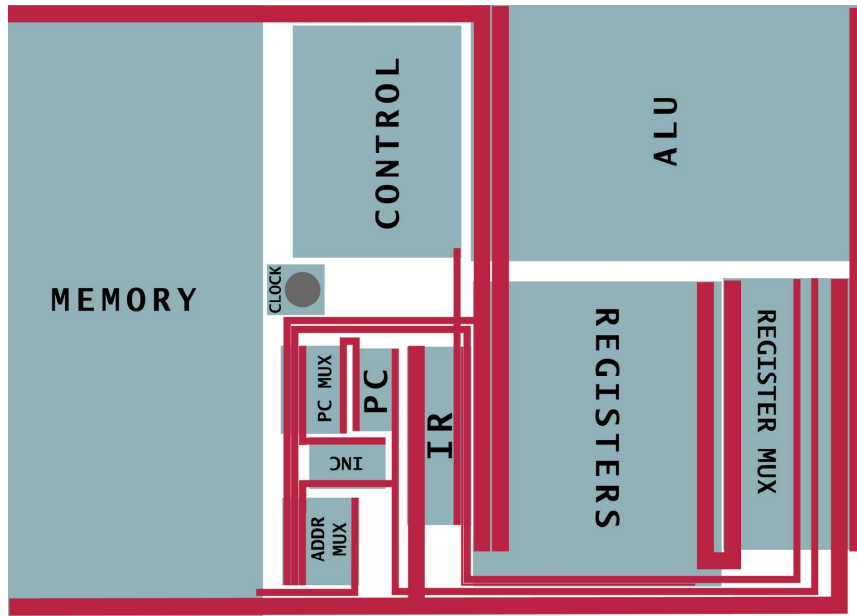
59

Build a TOY-Lite: Layout



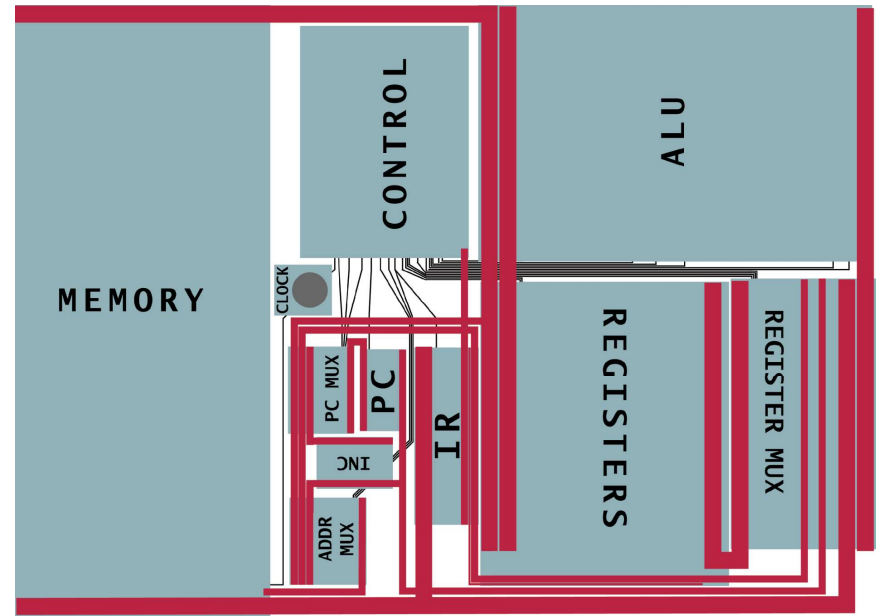
60

Build a TOY-Lite: Datapath

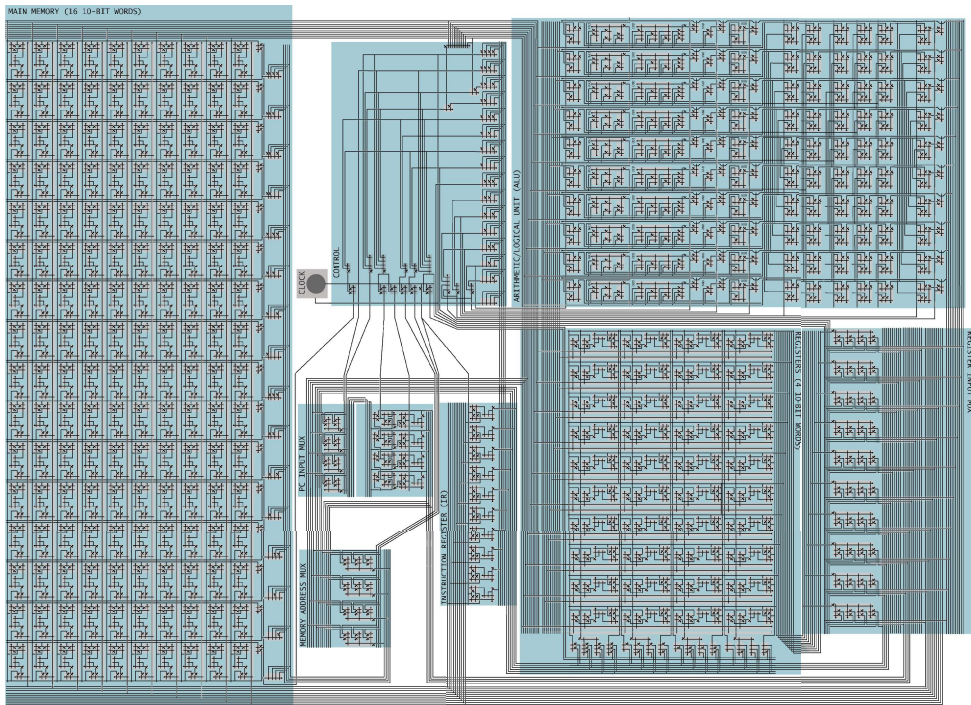


61

Build a TOY-Lite: Control

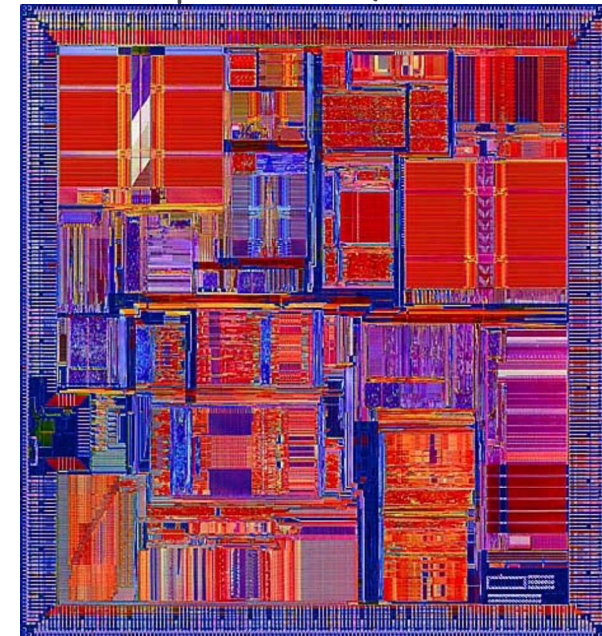


62



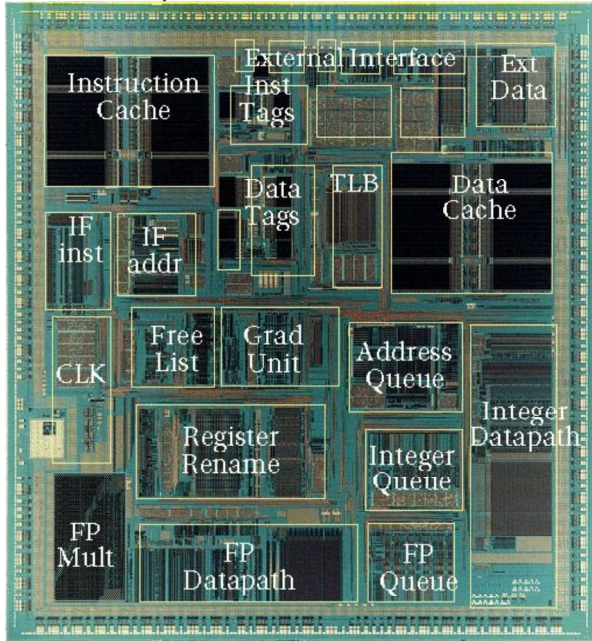
63

Real Microprocessor (MIPS R10000)



64

Real Microprocessor (MIPS R10000)



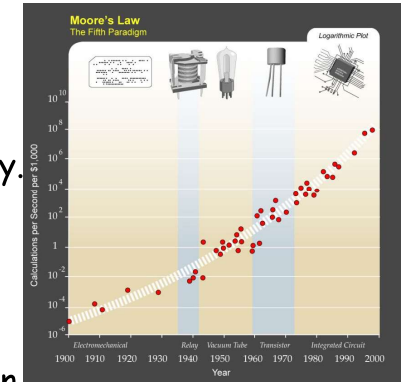
History + Future

Computer constructed by layering abstractions.

- Better implementation at low levels improves **everything**.
- Ongoing search for better abstract switch!

History.

- 1820s: mechanical switches.
- 1940s: relays, vacuum tubes.
- 1950s: transistor, core memory.
- 1960s: integrated circuit.
- 1970s: microprocessor.
- 1980s: VLSI.
- 1990s: integrated systems.
- 2000s: web computer.
- Future: quantum, optical soliton, ...



Ray Kurzweil (<http://en.wikipedia.org/wiki/Image:PPTMooreLawai.jpg>)