Microarchitecture trace: precise sequence of execution with values

Microarchitecture trace
4 lines for each cycle

Line 1: Cycle #

Line 2: State number (in binary)

Line 3: Relevant control signals

Line 4:

- Auxiliary register name and value (if any written);

  OR

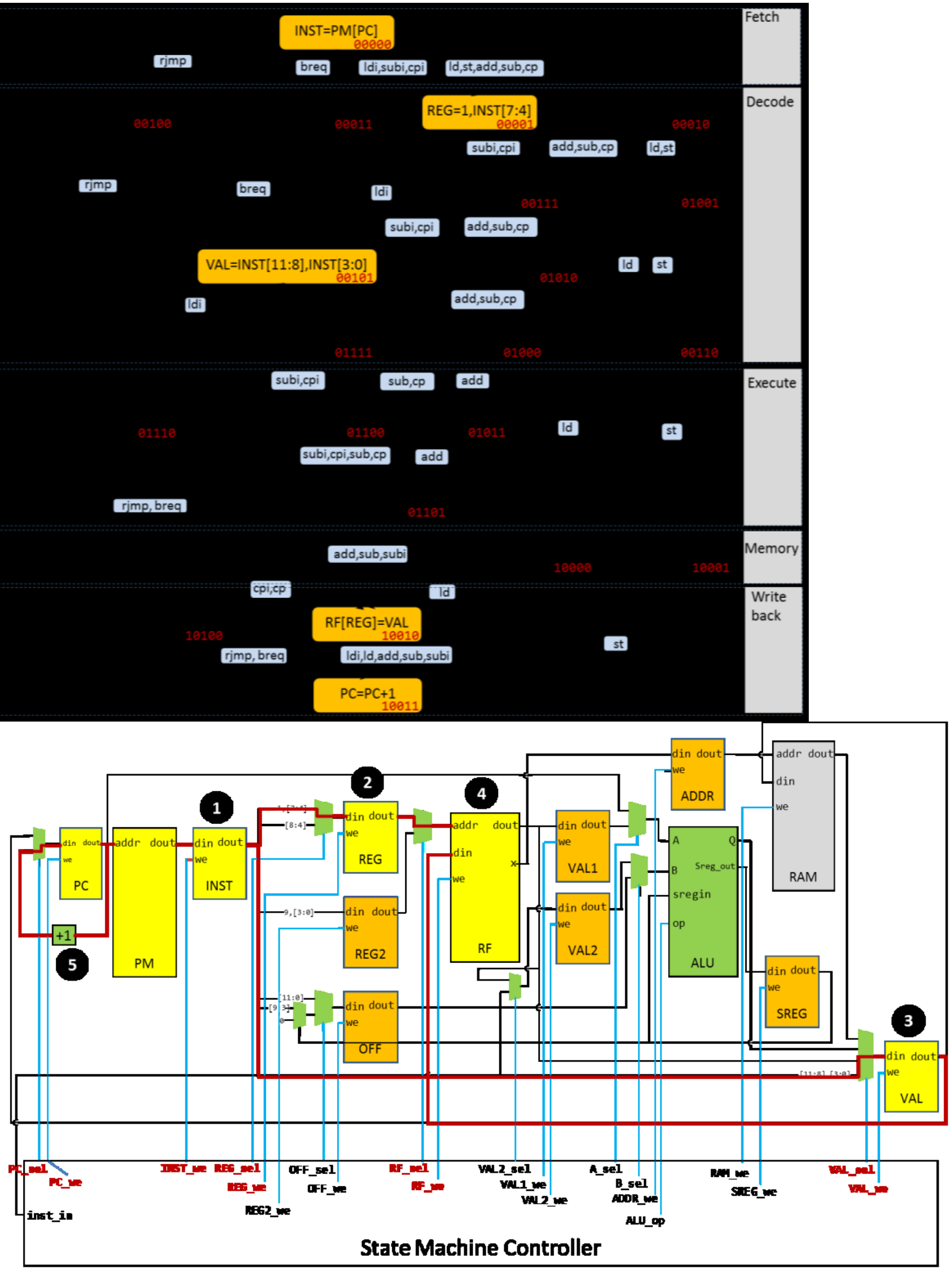
- Register filename number = value RF[#] = value

  OR

- PC = new value

```
ldi r16, 45
ldi r17, 23
ldi r18, 11
add r17, r18
breq 4
rjmp -3
```

ldi r16, 45



Cycle: 5

State: **10011 (PC = PC+1)**

Control signals: **PC_we = 1, PC_sel = 0, All others 0**

Aux registers: **None ; PC=1**

Cycle: 4

State: **10010 (RF[REG] = VAL)**

Control signals: **RF_we = 1, All others 0**

Aux registers: **None; RF[16]=45**

Cycle: 3

State: **00101 (VAL = INST[11:8],INST[3:0])**

Control signals: **VAL_we = 1, VAL_sel = 3, All others 0**

Aux registers: **VAL = 45**

Cycle: 2

State: **00001 (REG = 1,INST[7:4])**

Control signals: **REG_we = 1, REG_sel = 0, All others 0**

Aux registers: **REG = 16**

Cycle: 1

State: **00000 (INST = PM[PC])**

Control signals: **INST_we = 1, All others 0**

Aux registers: **INST = 57869**

Cycle: #

State:

Control signals:

Aux registers:

```
ldi r16, 45
ldi r17, 23
ldi r18, 11
add r17, r18
breq 4
rjmp -3
```

Cycle: 5
State: **10011 (PC = PC+1)**
Control signals: **PC_we = 1, PC_sel = 0, All others 0**
Aux registers: **None ; PC=2**

Cycle: 4
State: **10010 (RF[REG] = VAL)**
Control signals: **RF_we = 1, All others 0**
Aux registers: **None; RF[16]=23**

Cycle: 3
State: **00101 (VAL = INST[11:8],INST[3:0])**
Control signals: **VAL_we = 1, VAL_sel = 3, All others 0**
Aux registers: **VAL = 23**

Cycle: 2
State: **00001 (REG = 1,INST[7:4])**
Control signals: **REG_we = 1, REG_sel = 0, All others 0**
Aux registers: **REG = 17**

Cycle: 1
State: **00000 (INST = PM[PC])**
Control signals: **INST_we = 1, All others 0**
Aux registers: **INST = 57263**

```
ldi r16, 45
ldi r17, 23
ldi r18, 11
add r17, r18
breq 4
rjmp -3
```

Cycle: 5
State: **10011 (PC = PC+1)**
Control signals: **PC_we = 1, PC_sel = 0, All others 0**
Aux registers: **None ; PC=3**

Cycle: 4
State: **10010 (RF[REG] = VAL)**
Control signals: **RF_we = 1, All others 0**
Aux registers: **None; RF[18]=11**

Cycle: 3
State: **00101 (VAL = INST[11:8],INST[3:0])**
Control signals: **VAL_we = 1, VAL_sel = 3, All others 0**
Aux registers: **VAL = 11**

Cycle: 2
State: **00001 (REG = 1,INST[7:4])**
Control signals: **REG_we = 1, REG_sel = 0, All others 0**
Aux registers: **REG = 18**

Cycle: 1
State: **00000 (INST = PM[PC])**
Control signals: **INST_we = 1, All others 0**
Aux registers: **INST = 57387**

```
ldi r16, 45
ldi r17, 23
ldi r18, 11
add r17, r18
breq 4
rjmp -3
```

Cycle: 4
State: 01010 (REG2 = INST[9],INST[3:0])
Control signals: REG2_we = 1, All others 0
Aux registers: REG2 = 18

Cycle: 3
State: 00111 (VAL1=RF[REG])
Control signals: VAL1_we=1, RF_sel=0, All others = 0
Aux registers: VAL1=23

Cycle: 2
State: 00010 (REG=INST[8:4])
Control signals: REG_we=1, REG_sel = 1, All others = 0
Aux registers: REG=17

Cycle: 1
State: 00000 (INST=PM[PC])
Control signals:INST_we=1
Aux registers:INST=3858 which is same as 0000111100010010

Cycle: #
State: ()
Control signals:
Aux registers:

```
ldi r16, 45
ldi r17, 23
ldi r18, 11
add r17, r18
breq 4
rjmp -3
```

Cycle: 9
State: 10011 (PC = PC+1)
Control signals: PC_we = 1, PC_sel = 0, All others 0
Aux registers: PC=4

Cycle: 8
State: 10010 (RF[REG] = VAL)
Control signals: RF_we = 1,RF_sel = 0, All others 0
Aux registers: None
Register file: RF[17] = 44

Cycle: 7
State: 01101 (Update SREG)
Control signals: SREG_we = 1, All others 0
Aux registers: SREG (ZFLAG = 0)

Cycle: 6
State: 01011 (VAL = VAL1 + VAL2)
Control signals: VAL_we = 1, VAL_sel = 1, A_sel = 1, B_sel = 0, ALU_op = 0,All others 0
Aux registers: VAL = 44

Cycle: 5
State: 01000 (VAL2 = RF[REG2])
Control signals: VAL2_we = 1, VAL2_sel = 0, RF_sel = 1, All others 0
Aux registers: VAL2 = 11

Trace simulator here:
http://discovering.cs.wisc.edu/sim/uarch/uarch.html

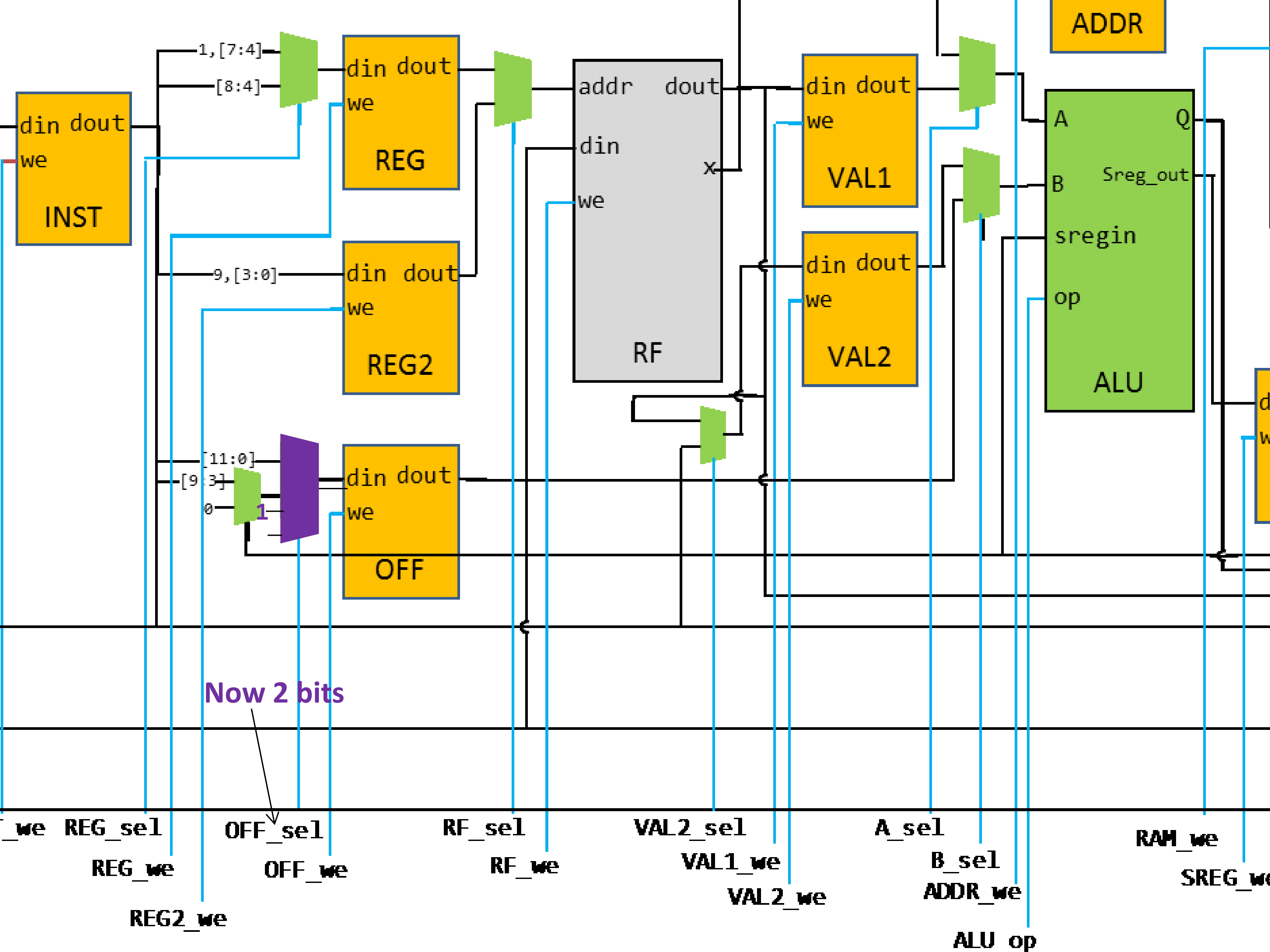
Adding a new instruction!

- Figure out whether current states are enough

- Implementation
  - Need new auxiliary registers?
  - Need new circuit blocks?
  - Need new muxes?
  - Need new control signals?

inc (instruction)

- inc r1
  - Put name in REG (state: 00010)
  - Read RF[REG] and put into VAL1 (state: 00111)
  - Put 1 into B somehow!
    - By putting 1 into OFF **(new state)**
  - Perform ALU_op = 1 (addition), VAL=VAL1+OFF **(new state)**
  - Write VAL into REG (state: 10010)
  - PC=PC+1 (state: 10011)



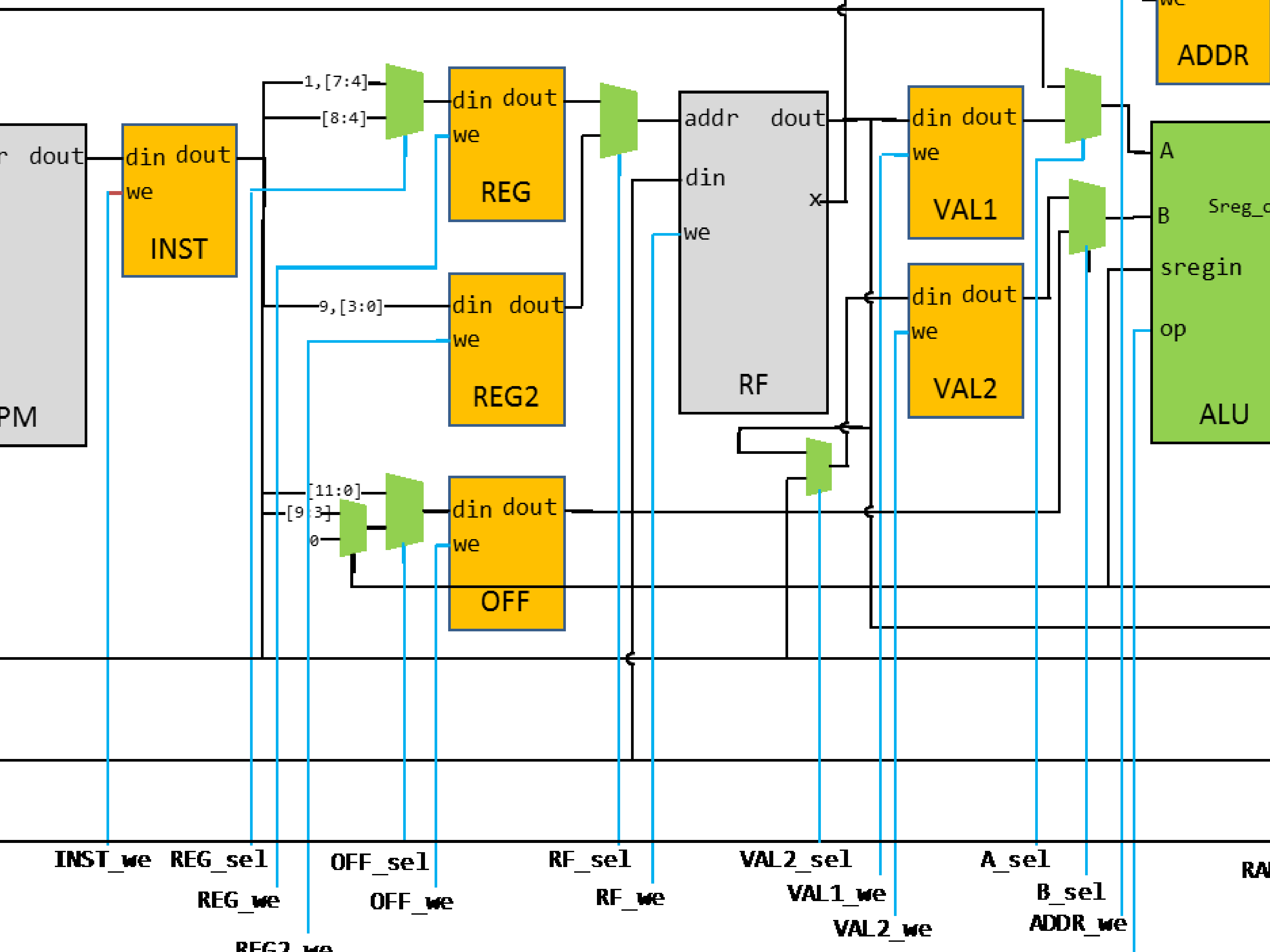
din dout
we
INST
1,[7:4]
[8:4]
din dout
we
REG
9,[3:0]
din dout
we
REG2
[11:0]
[9:3]
0
1
din dout
we
OFF
addr dout
din
we
x
RF
din dout
we
VAL1
din dout
we
VAL2
ADDR
A
B
sregin
op
Q
Sreg_out
ALU
Now 2 bits
_we
REG_sel
REG_we
REG2_we
OFF_sel
OFF_we
RF_sel
RF_we
VAL2_sel
VAL1_we
VAL2_we
A_sel
B_sel
ADDR_we
ALU op
RAM_we
SREG_we

regjump instruction

- regjump R#; PC = PC + RF[R#] + 1
  R# in INST[9],INST[3:0]
  - INST=PM[PC] (state: 00000)
  - Put R# in REG2 (state: 01010)
  - Read RF[REG2] and put into VAL2 (state: 01000)
  - VAL=PC+VAL2 **(new state: A_sel=0, B_sel=0)**
  - PC=VAL (state: 10100)
  - PC=PC+1

brlo, brsh, brne

- No new state necessary!

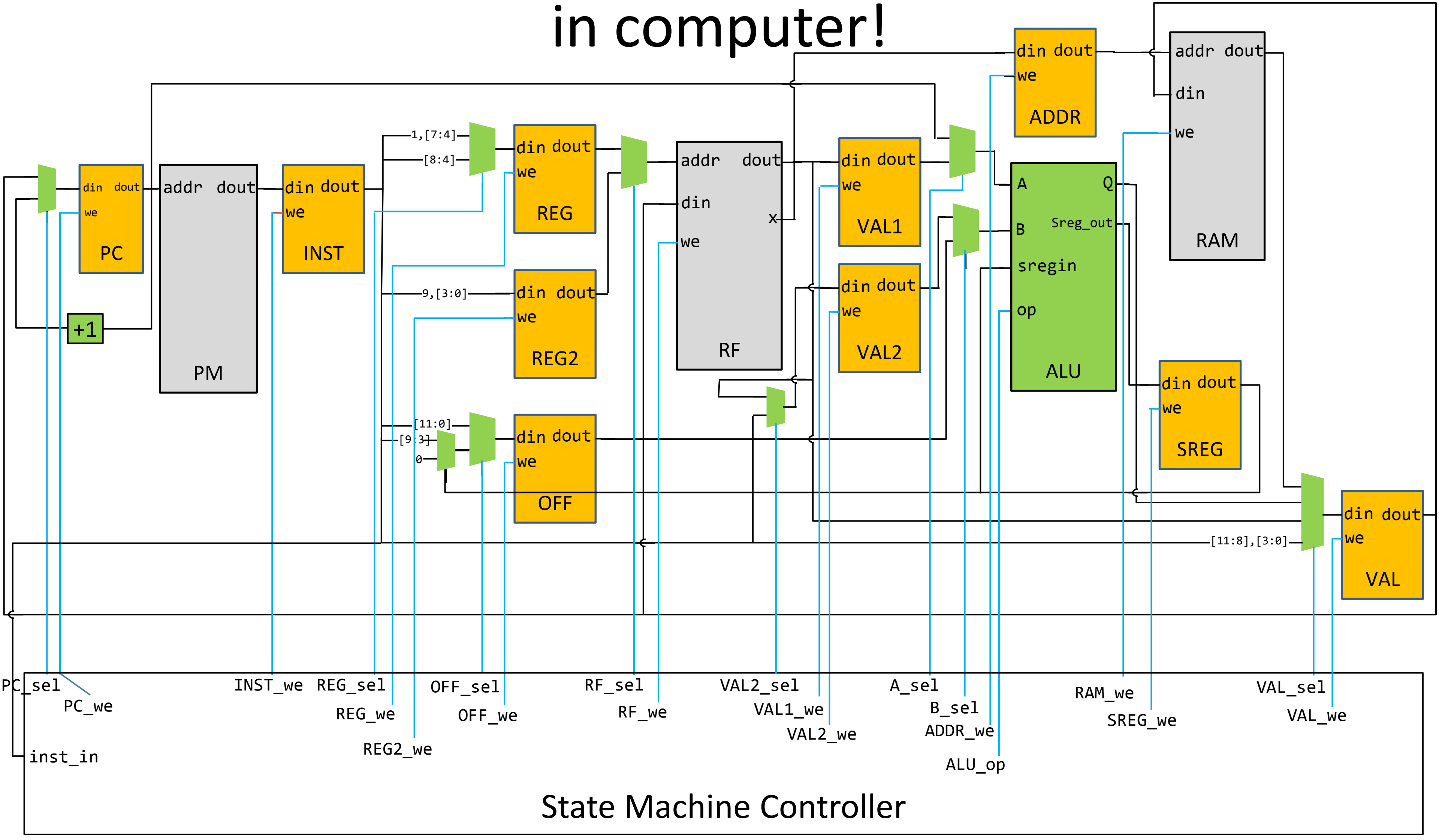

ADDR
dout
din
dout
we
INST
PM
1,[7:4]
[8:4]
din dout
we
REG
9,[3:0]
din dout
we
REG2
[11:0]
[9:3]
0
din dout
we
OFF
addr
dout
din
x
we
RF
din dout
we
VAL1
din dout
we
VAL2
A
B
Sreg_
sregin
op
ALU
INST_we
REG_sel
REG_we
REG2_we
OFF_sel
OFF_we
RF_sel
RF_we
VAL2_sel
VAL1_we
VAL2_we
A_sel
B_sel
ADDR_we
RA

Logic

C	Z	Select
0	0	1
0	1	0
1	0	0

One IMPORTANT take-away

Every module implements its interface and thus becomes independent of anything else happing in computer!



Chapter summary

- State machine, circuit blocks, registers
- Processor:
  - 5 execution steps
  - Implemented with microarchitecture
    - Datapath
    - Control signals (state-machine controller)
  - Conceptual execution
    - Traversal of set of states

What I am going to test you on

- Basic understanding of circuit blocks
- Combination of VERY SIMPLE circuits
- Execution trace of instructions
  - State transitions
  - Microarchitecture state
- Given a **new** instruction,
  - How to modify state machine?
  - Vague ideas on how to modify processor.

Cycles per instruction (CPI)

Instruction type	Cycles
ldi	5
subi	8
cpi	7
ld	6
st	6
add	9
sub	9
Cp	8
breq	5
rjmp	5

Iron Law of Computing

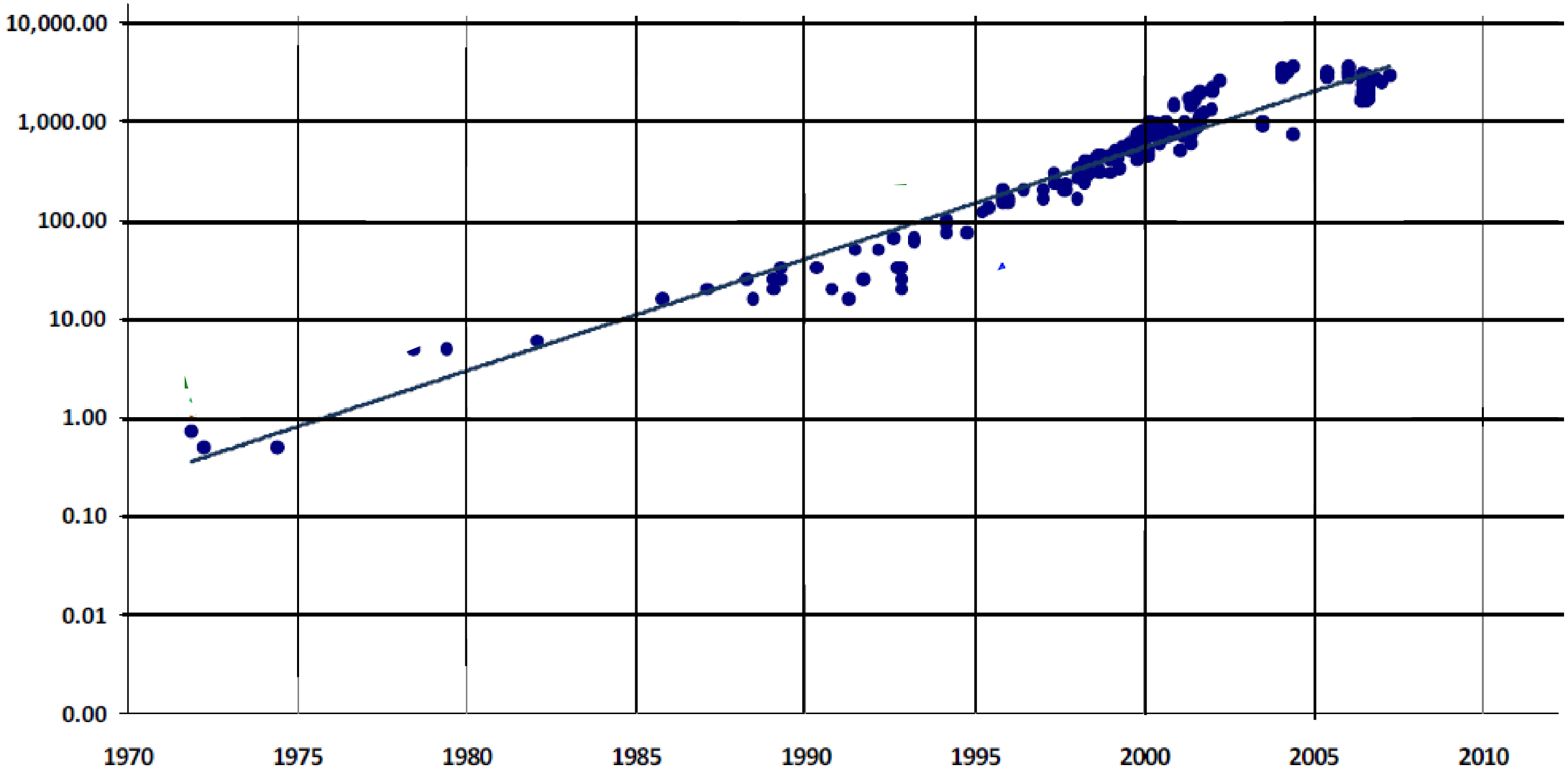
$$Execution\ Time = \left(\sum_{i=0}^{9} CPI_i * \#instructions_i\right) * Clock\ Period$$

Big ideas

$$Execution\ Time = \left(\sum_{i=0}^{9} CPI_i * \#instructions_i \right) * Clock\ Period$$

1. Pipelining (reduce clock period)
2. Caching
3. Superscalar execution
4. Out of order execution
5. Speculation

5 orders of magnitude!



11/13

Microarchitecture trace: tracing thru at a level of detail beyond the ISA. Text-representation of everything that happens inside the machine. Shows aux registers, state, relevant control signals.

RF[#] for a store instruction → writing to memory

You should be able to look at a program and tell (with a little bit of math) how many cycles these instructions should take.

Ex: ldi takes five instructions, 5 * 3 = 15
How can you determine how many lines an instruction will take? Go through the state machine and trace it. Each state is a cycle!

Execution of one instruction is independent from the ones preceding it.

(The diagrams start at cycle 1.)

What methodology was used to assign the state numbers? Short answer: random.

How do we calculate INST?
1. Look up the encoding for LDI from chapter 6.
2. Convert all respective values (register, immediate) into binary and place them into the encoding.
3. Convert your binary string to decimal/hex.

(In the below table you can assume that all other control signals = 0)

1	Ldi	10000	INST_we = 1, INST = 57869 (use chapter 6 opcodes)
2	Ldi	00001	REG_we =1, REG_sel = 0, REG = 16
3	Ldi	00101	VAL_we = 1, VAL_sel = 3, VAL = 45
4	ldi	10010	RF_we = 1, RF_sel = 0, RF[16] = 45
5	ldi	10011	PC_sel = 1, PC_we = 1, PC = 1
--------*END INSTRUCTION 1, BEGIN INSTRUCTION 2*			
(etc)			

Also check out the awesome simulator by Daniel! → discovering.cs.wisc.edu/sim/uarch/uarch.html

And now for something different...

How would you add a new instruction to a machine? Four-step process on slides.

IMPORTANT: note that every module works independently of everything else! They all just use whatever in/output they get without caring of pieces before and after.

How to reduce computation time required?
1. Pipelining: have multiple instructions running at once. Concurrent execution reduces time required. Reduces clock period, increases clock frequency. Today we have pipelines as long as 25 stages.
2. Caching: You actually can't access memory in a single cycle. What you do instead is store a very small, quick memory with relevant values before you actually know what they are. (Magic.)

3. Superscalar execution: multiple ALUs! Having four means you can execute four instructions at a time. That makes you four times faster. Your architecture becomes bigger but transistors are becoming smaller as time goes on.
4. Out of order execution: given instructions, sometimes you can execute in a different order than the programmer provided. Pretty bizarre, eh?
5. Speculation: constant prediction of what's going to happen in your microarchitecture and hoping that the prediction was right.

All of these have helped increase performance by FIVE ORDERS OF MAGNITUDE!