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

ldi r17, 23
add r17, r18

breq 4
State: 10010 (RF[REG] = VAL)
Control signals: RF_we = 1, RF_sel = 0, All others 0
Aux registers: None
Register file: RF[17] = 44

rjmp -3
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)
Now 2 bits
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!
## Logic

<table>
<thead>
<tr>
<th>C</th>
<th>Z</th>
<th>Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
One IMPORTANT take-away
Every module implements its interface and thus becomes independent of anything else happening 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)

<table>
<thead>
<tr>
<th>Instruction type</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldi</td>
<td>5</td>
</tr>
<tr>
<td>subi</td>
<td>8</td>
</tr>
<tr>
<td>cpi</td>
<td>7</td>
</tr>
<tr>
<td>ld</td>
<td>6</td>
</tr>
<tr>
<td>st</td>
<td>6</td>
</tr>
<tr>
<td>add</td>
<td>9</td>
</tr>
<tr>
<td>sub</td>
<td>9</td>
</tr>
<tr>
<td>Cp</td>
<td>8</td>
</tr>
<tr>
<td>breq</td>
<td>5</td>
</tr>
<tr>
<td>rjmp</td>
<td>5</td>
</tr>
</tbody>
</table>
Iron Law of Computing

\[ \text{Execution Time} = \left( \sum_{i=0}^{9} CPI_i \ast \text{instructions}_i \right) \ast \text{Clock Period} \]
Big ideas

$$\text{Execution Time} = \left( \sum_{i=0}^{9} CPI_i \times \#\text{instructions}_i \right) \times \text{Clock Period}$$

1. Pipelining (reduce clock period)
2. Caching
3. Superscalar execution
4. Out of order execution
5. Speculation
5 orders of magnitude!
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, $5 \times 3 = 15$

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)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ldi</td>
<td>10000</td>
<td>INST_we = 1, INST = 57869 (use chapter 6 opcodes)</td>
</tr>
<tr>
<td>2</td>
<td>Ldi</td>
<td>00001</td>
<td>REG_we =1, REG_sel = 0, REG = 16</td>
</tr>
<tr>
<td>3</td>
<td>Ldi</td>
<td>00101</td>
<td>VAL_we = 1, VAL_sel = 3, VAL = 45</td>
</tr>
<tr>
<td>4</td>
<td>ldi</td>
<td>10010</td>
<td>RF_we = 1, RF_sel = 0, RF[16] = 45</td>
</tr>
<tr>
<td>5</td>
<td>ldi</td>
<td>10011</td>
<td>PC_sel = 1, PC_we = 1, PC = 1</td>
</tr>
</tbody>
</table>

--------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!