More Sequential Circuits, plus Architecture
Sequential vs. Combinational Circuits

Combinational circuits.
- Output determined solely by inputs.
- Can draw solely with left-to-right signal paths.

Sequential circuits.
- Output determined by inputs AND previous outputs.
- Feedback loop.
SR Flip-Flop

SR Flip-Flop.
- $S = 1, R = 0$ (set) $\Rightarrow$ “Flips” bit on.
- $S = 0, R = 1$ (reset) $\Rightarrow$ “Flops” bit off.
- $S = R = 0$ $\Rightarrow$ Status quo.
- $S = R = 1$ $\Rightarrow$ Not allowed.

Implementation

Interface
Clocked SR Flip-Flop

Clocked SR Flip-Flop.
- Same as SR flip-flop except $S$ and $R$ only active when clock is 1.

![Implementation diagram](image)

![Interface diagram](image)
Clocked D Flip-Flop

- Output follows D input while clock is 1.
- Output is remembered while clock is 0.

Implementation

![Diagram of Clocked SR flip flop and Clocked D flip flop](image)

Interface

Q  
clk  
D  

Q  
clk  
D  

Waveforms for Q, clk, and D
Memory Overview

Computers and TOY have many types of memory.
  ■ Program counter.
  ■ Registers.
  ■ Main memory.

We implement each bit of memory with a clocked D flip-flop.

Need mechanism to organize and manipulate GROUPS of related bits.
  ■ TOY has 16-bit words.
  ■ Memory hierarchy makes architecture manageable.
Bus

16-bit bus.
- Bundle of 16 wires.
- Memory transfer, register transfer.

8-bit bus.
- Bundle of 8 wires.
- TOY memory address.

4-bit bus.
- Bundle of 4 wires.
- TOY register address.
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.
Register File Interface

n-by-k register file.
- Bank of \( n \) registers; each stores \( k \) bits.
- Read and write information to \textit{one} of \( n \) registers.
  - \( \log_2 n \) address inputs specifies which one
- Addressed bits always appear on output.
- If write enable and clock are asserted, \( k \) input bits are copied into addressed register.

Examples.
- TOY registers: \( n = 16, k = 16 \).
- TOY main memory: \( n = 256, k = 16 \).
- Real computer: \( n = 256 \) million, \( k = 32 \).
  - 1 GB memory
  - (1 Byte = 8 bits)
Implementation example: TOY main memory.
- Use 256 16-bit registers.
- Multiplexer and decoder are combinational circuits.
Register File Implementation: Reading

Implementation example: TOY main memory.
- Use 256 16-bit registers.
- Multiplexer is combinational circuit.
$2^n$-to-1 Multiplexer

- $n$ select inputs, $2^n$ data inputs, 1 output.
- Copies "selected" data input bit to output.

$n = 8$ for main memory

8-to-1 Mux Interface

8-to-1 Mux Implementation
$2^n$-to-1 Multiplexer

- **$2^n$-to-1 multiplexer.**
  - $n$ select inputs, $2^n$ data inputs, 1 output.
  - Copies "selected" data input bit to output.

$n = 8$ for main memory

8-to-1 Mux Interface

8-to-1 Mux Implementation
$2^n$-to-1 Multiplexer, Width = $k$

$n = 8$, $k = 16$ for main memory

$2^n$-to-1 multiplexer, width = $k$.
- Select from one of $2^n$ $k$-bit buses.
- Copies $k$ "selected" data bits to output.
- Layering $k$ $2^n$-to-1 multiplexers.

Interface for 2-to-1 MUX, width = 4

Implementation for 2-to-1 MUX, width = 4

4 copies of same signal
Register File Implementation: Writing

Implementation example: TOY main memory.
- Use 256 16-bit registers.
- Decoder is combinational circuit.
n-Bit Decoder

- $n$ address inputs, $2^n$ data outputs.
- Addressed output bit is 1; others are 0.

$n = 8$ for main memory

### 3-Bit Decoder Implementation

#### 3-Bit Decoder Interface

```
000  x_0
001  x_1
010  x_2
011  x_3
100  x_4
101  x_5
110  x_6
111  x_7
```

#### 3-Bit Decoder Implementation

- $s_2$, $s_1$, $s_0$ select

- $x_0$, $x_1$, $x_2$, $x_3$, $x_4$, $x_5$, $x_6$, $x_7$ outputs
### n-Bit Decoder

**n-bit decoder.**
- \( n \) address inputs, \( 2^n \) data outputs.
- Addressed output bit is 1; others are 0.

#### 3-Bit Decoder

- **3-Bit Decoder Interface**
- **3-Bit Decoder Implementation**

\( n = 8 \) for main memory
Implementation example: TOY main memory.

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

Read address can be different from Write address
- Not in Main Memory (one address from instruction or PC)
- But definitely in TOY registers (read from and write to different registers)

Can have multiple “ports”
- TOY registers supply TWO values per instruction
- How? Just get another set of 16-to-1, 16-wide multiplexors
  (and one more 4-bit address)

Actual technologies for register and memory are different.
- Register files are relatively small and very fast (expensive per bit)
- Memories are relatively large and pretty fast (very cheap per bit)
- Drastic evolution of technology over time (Moore’s Law)
6.3: TOY Machine Architecture
The TOY Machine

TOY machine.
- 256 16-bit words of memory.
- 16 16-bit registers.
- 1 8-bit program counter.
- 16 instructions types.

What we've done.
- Written programs for the TOY machine.
- Software implementation of fetch-execute cycle.
  - TOY simulator.

Our goal today.
- Hardware implementation of fetch-execute cycle.
  - TOY computer.
Designing a Processor

How to build a microprocessor?

- 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

- Establish clocking methodology.
  - 2-cycle design: fetch, execute

- Analyze how to implement each instruction.
  - determine settings of control signals
Instruction Set Architecture

Instruction set architecture (ISA).

- 16-bit words, 256 words of memory, 16 registers.
- Determine set of primitive instructions.
  - too narrow ⇒ cumbersome to program
  - too broad ⇒ cumbersome to build hardware
- TOY machine: 16 instructions.

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: halt</td>
<td>8: load</td>
</tr>
<tr>
<td>1: add</td>
<td>9: store</td>
</tr>
<tr>
<td>2: subtract</td>
<td>A: load indirect</td>
</tr>
<tr>
<td>3: and</td>
<td>B: store indirect</td>
</tr>
<tr>
<td>4: xor</td>
<td>C: branch zero</td>
</tr>
<tr>
<td>5: shift left</td>
<td>D: branch positive</td>
</tr>
<tr>
<td>6: shift right</td>
<td>E: jump register</td>
</tr>
<tr>
<td>7: load address</td>
<td>F: jump and link</td>
</tr>
</tbody>
</table>
Designing a Processor

How to build a microprocessor?

- 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

- Establish clocking methodology.
  - 2-cycle design: fetch, execute

- Analyze how to implement each instruction.
  - determine settings of control signals
Arithmetic Logic Unit

TOY ALU.
- Big combinational circuit.
- 16-bit buses for inputs and output.
- Add, subtract, and, xor, shift left, shift right, copy input 2.

<table>
<thead>
<tr>
<th>op</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, -</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&amp;</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>^</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>&lt;&lt;, &gt;&gt;</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>input 2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Arithmetic Logic Unit: Implementation

<table>
<thead>
<tr>
<th>op</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, -</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&amp;</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>^</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>&lt;&lt;, &gt;&gt;</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>input 2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Main Memory

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

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>0A</th>
<th>0B</th>
<th>0C</th>
<th>0D</th>
<th>0E</th>
<th>0F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>1A</td>
<td>1B</td>
<td>1C</td>
<td>1D</td>
<td>1E</td>
<td>1F</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>2A</td>
<td>2B</td>
<td>2C</td>
<td>2D</td>
<td>2E</td>
<td>2F</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>3A</td>
<td>3B</td>
<td>3C</td>
<td>3D</td>
<td>3E</td>
<td>3F</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>4A</td>
<td>4B</td>
<td>4C</td>
<td>4D</td>
<td>4E</td>
<td>4F</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>5A</td>
<td>5B</td>
<td>5C</td>
<td>5D</td>
<td>5E</td>
<td>5F</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>6A</td>
<td>6B</td>
<td>6C</td>
<td>6D</td>
<td>6E</td>
<td>6F</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>7A</td>
<td>7B</td>
<td>7C</td>
<td>7D</td>
<td>7E</td>
<td>7F</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td>8A</td>
<td>8B</td>
<td>8C</td>
<td>8D</td>
<td>8E</td>
<td>8F</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>9A</td>
<td>9B</td>
<td>9C</td>
<td>9D</td>
<td>9E</td>
<td>9F</td>
</tr>
<tr>
<td></td>
<td>A0</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>A4</td>
<td>A5</td>
<td>A6</td>
<td>A7</td>
<td>A8</td>
<td>A9</td>
<td>AA</td>
<td>AB</td>
<td>AC</td>
<td>AD</td>
<td>AE</td>
<td>AF</td>
</tr>
<tr>
<td></td>
<td>B0</td>
<td>B1</td>
<td>B2</td>
<td>B3</td>
<td>B4</td>
<td>B5</td>
<td>B6</td>
<td>B7</td>
<td>B8</td>
<td>B9</td>
<td>BA</td>
<td>BB</td>
<td>BC</td>
<td>BD</td>
<td>BE</td>
<td>BF</td>
</tr>
<tr>
<td></td>
<td>C0</td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td>C4</td>
<td>C5</td>
<td>C6</td>
<td>C7</td>
<td>C8</td>
<td>C9</td>
<td>CA</td>
<td>CB</td>
<td>CC</td>
<td>CD</td>
<td>CE</td>
<td>CF</td>
</tr>
<tr>
<td></td>
<td>D0</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D8</td>
<td>D9</td>
<td>DA</td>
<td>DB</td>
<td>DC</td>
<td>DD</td>
<td>DE</td>
<td>EF</td>
</tr>
<tr>
<td></td>
<td>E0</td>
<td>E1</td>
<td>E2</td>
<td>E3</td>
<td>E4</td>
<td>E5</td>
<td>E6</td>
<td>E7</td>
<td>E8</td>
<td>E9</td>
<td>EA</td>
<td>EB</td>
<td>EC</td>
<td>ED</td>
<td>EE</td>
<td>FE</td>
</tr>
<tr>
<td></td>
<td>F0</td>
<td>F1</td>
<td>F2</td>
<td>F3</td>
<td>F4</td>
<td>F5</td>
<td>F6</td>
<td>F7</td>
<td>F8</td>
<td>F9</td>
<td>FA</td>
<td>FB</td>
<td>FC</td>
<td>FD</td>
<td>FE</td>
<td>00</td>
</tr>
</tbody>
</table>

### 16-bit Write Data

### 16-bit Read Data

### 8-bit Address

### Cl Write
 Registers

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: \( R_1 \leftarrow R_2 + R_3 \).
- 3 address inputs, 1 data input, 2 data outputs.
- Add decoders and muxes for additional ports.
How to build a microprocessor?

- 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

- Establish clocking methodology.
  - 2-cycle design: fetch, execute

- Analyze how to implement each instruction.
  - determine settings of control signals
Datapath and Control

Datapath.
- Layout and interconnection of components.
- Must accommodate all instruction types.

Control.
- Choreographs the "flow" of information on the datapath.
- Depending on instruction, different control wires are turned on.

![Diagram of MUX and PC with control wires and datapath wires indicating flow of information](image-url)
Datapath and Control

Datapath.
- Layout and interconnection of components.
- Must accommodate all instruction types.

Control.
- Choreographs the "flow" of information on the datapath.
- Depending on instruction, different control wires are turned on.
Real Microprocessor Chip (Intel Core i7)
The TOY Datapath

PC: pc for branch, jump

Memory: addr for loads, stores

IR: pc for jal

Cond Eval: result of arithmetic, logic, or addr for load addr

Registers: pc for branch, jump

ALU: addr for loads, stores

W Data: pc + 1

R Data: store data
The TOY Datapath: Add

Before fetch:
\[ pc = 20, \text{mem}[20] = 1234 \]

After fetch:
\[ pc = 21 \]
The TOY Datapath: Add

Before execute:

- \( pc = 21 \)

After execute:

- \( pc = 21 \)
- \( R[2] = 008C \)
Do Try This At Home

Trace the flow of some other instructions through the datapath picture.
Designing a Processor

How to build a microprocessor?

- 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

- Establish clocking methodology.
  - 2-cycle design: fetch, execute

- Analyze how to implement each instruction.
  - determine settings of control signals
Clocking Methodology

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.
Clocking Methodology

4 distinguishable epochs.
- During fetch phase.
- At very end of fetch phase.
- During execute phase.
- At very end of execute phase.

Ex: can only write at very end of execute phase.
- \( R1 \leftarrow R1 + R1 \)

![Clocking Methodology Diagram]

**Clocking Methodology Diagram**

- **Fetch**
- **Execute**
- **Clock**
- **Time for one instruction**

**Registers**
- W Data
- W Addr
- A Addr
- A Data
- B Addr
- B Data
Designing a Processor

How to build a microprocessor?

- 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

- Establish clocking methodology.
  - 2-cycle design: fetch, execute

- Analyze how to implement each instruction.
  - determine settings of control signals
Control

Control: controls components, enables connections.

- Input: opcode, clock, conditional evaluation. (green)
- Output: control wires. (orange)
Control

Control: controls components, enables connections.

- Input: opcode, clock, conditional evaluation. (green)
- Output: control wires. (orange)
Implementation of Control

4-BIT DECODER

opcode

Inputs

WRITE MEM
WRITE IR
ALU SELECT 0
ALU MUX
READ REG A MUX

plus a few more
Implementation of Control: Store

4-BIT DECODER

- halt
- add
- subtract
- and
- xor
- shift left
- shift right
- load addr
- load
- store
- store indirect
- load indirect
- branch pos
- branch zero
- jump reg
- jump + link
- cond positive
- cond zero
- fetch
- execute
- clock

Inputs

- WRITE MEM
- WRITE IR
- ALU SELECT 0
- ALU MUX
- READ REG A MUX

plus a few more
Control: Execute Phase of Store
Stand-Alone Registers

Instruction Register

Program Counter

- IR: Instruction Register
- Clock
- Fetch

- PC: Program Counter
- Clock
- Jump, Branch
- Execute
- Jump, Link
- Jump, Reg
- Bzero, Bpos
- Bpos > 0
- Bpos = 0
- PC + 1
Pipelining

- At any instant, processor is either fetching instructions or executing them (and so half of circuitry is idle).
- Why not fetch next instruction while current instruction is executing?
  - Analogy: washer / dryer.

Issues.

- Jump and branch instructions change PC.
  - "Prefetch" next instruction.
- Fetch and execute cycles may need to access same memory.
  - Solution: use two memory “caches”.

Result.

- Better utilization of hardware.
- Can double speed of processor.
All three of our logic primitives can be made using a *single* type of electronic primitive: the transistor!