Instructions

Philipp Koehn

16 February 2018
number adder
Design Goal

- Build a machine that adds several numbers together
- Numbers stored in 64 KB RAM
- Idea: Loop through memory with ripple counter
- Read/write 8 bits at a time (one byte)
- 16 bit address space: \(2^{16}=65,536\) bytes

64 KB RAM
Control Panel

64-KB RAM Control Panel

A_{15} A_{14} A_{13} A_{12} A_{11} A_{10} A_{9} A_{8} A_{7} A_{6} A_{5} A_{4} A_{3} A_{2} A_{1} A_{0}

D_{7} D_{6} D_{5} D_{4} D_{3} D_{2} D_{1} D_{0}

1 1 0 0

Write Takeover
We can enter numbers and inspect with a control panel.
Ripple Counter

CLK | 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
OUT1 | 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
OUT2 | 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
OUT3 | 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
• Ripple counter rotates through number 0, 1, ...

• Each clock cycle, a new number is emitted from memory
• Adds two numbers: S = A + B

• Overflow: Carry out (CO)
8-bit memory

Edge-triggered: stores value when clock turns to 1

To be used as accumulator
Connecting Adder and Latch

- Adder adds new value (DATA) to accumulator
- Edge trigger prevents immediate feedback
- Output (OUT) may be shown with light bulbs
Number Adder

- Everything put together
Halt

- Halt when external switch is turned on
- Or: cut connection to clock if ripple counter reaches final number
multiple operations
Modification: Alternate Add and Subtract

• Let’s say we want to alternate between adding and subtracting

• We already built an integrated adder and subtractor

• Idea: indicate operation from last bit of ripple counter
Alternate Add and Subtract

- Bit 0 of ripple counter as instruction flag
instructions
Goal

• Control operations by instructions stored in memory

⇒ A *programmable computer*

• First idea
  - separate instruction memory
  - instructions: add or subtract
Operation Codes for Instructions

- Each operation is encoded by a byte value

<table>
<thead>
<tr>
<th>Operation</th>
<th>Code (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>20h</td>
</tr>
<tr>
<td>Subtract</td>
<td>21h</td>
</tr>
</tbody>
</table>

- Example

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Data</th>
<th>Accumulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000h</td>
<td>20h</td>
<td>01h</td>
<td>01h</td>
</tr>
<tr>
<td>0001h</td>
<td>20h</td>
<td>02h</td>
<td>03h</td>
</tr>
<tr>
<td>0002h</td>
<td>21h</td>
<td>01h</td>
<td>02h</td>
</tr>
<tr>
<td>0003h</td>
<td>20h</td>
<td>08h</td>
<td>0ah</td>
</tr>
<tr>
<td>0004h</td>
<td>21h</td>
<td>03h</td>
<td>07h</td>
</tr>
</tbody>
</table>
More Instructions

- Load: load number from memory into accumulator
- Store: store accumulator value in memory
- Halt: block clock
Operation Codes for Instructions

- Each operation is encoded by a byte value

<table>
<thead>
<tr>
<th>Operation</th>
<th>Code (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>10h</td>
</tr>
<tr>
<td>Store</td>
<td>11h</td>
</tr>
<tr>
<td>Add</td>
<td>20h</td>
</tr>
<tr>
<td>Subtract</td>
<td>21h</td>
</tr>
<tr>
<td>Halt</td>
<td>FFh</td>
</tr>
</tbody>
</table>
Store

OSCILLATOR

AND

CLK

RIPPLE COUNTER

CONTROL PANEL

Data

HALT

Code

CTRL UNIT

64Kx8 RAM

CONTROL PANEL

8-BIT LATCH

D

Q

OUT

W

A

64Kx8 RAM

DI

SUB

HALT

LD

STR

INSTR

SUB ALU

CO S CI

SELECTOR

>CLK D

Q

Philipp Koehn

Computer Systems Fundamental: Instructions

16 February 2018
Halt
Each operation changes certain flags

<table>
<thead>
<tr>
<th>Operation</th>
<th>Code (hex)</th>
<th>LD</th>
<th>STR</th>
<th>SUB</th>
<th>HALT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>10h</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Store</td>
<td>11h</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Add</td>
<td>20h</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Subtract</td>
<td>21h</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Halt</td>
<td>FFh</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
## Program Example

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Data</th>
<th>Accumulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000h</td>
<td>10h</td>
<td>56h</td>
<td>56h</td>
</tr>
<tr>
<td>0001h</td>
<td>20h</td>
<td>2Ah</td>
<td>80h</td>
</tr>
<tr>
<td>0002h</td>
<td>21h</td>
<td>38h</td>
<td>48h</td>
</tr>
<tr>
<td>0003h</td>
<td>11h</td>
<td>Store</td>
<td>48h</td>
</tr>
<tr>
<td>0004h</td>
<td>FFh</td>
<td>Halt</td>
<td>48h</td>
</tr>
</tbody>
</table>
adding 16 bit numbers
Adding 16 Bit Numbers

• 1 byte integers will not suffice in practice
  – unsigned: 0 to 255
  – signed: -128 to 127

• Let’s use 2 bytes (16 bit)

• How can we do addition with our 8-bit adder?
  Add the bytes separately
Example

- Task: 76ABh + 232Ch

- Lower order byte
  \[\begin{array}{c}
  ABh \\
  +2Ch \\
  \hline
  D7h
  \end{array}\]

- Higher order byte
  \[\begin{array}{c}
  76h \\
  +23h \\
  \hline
  99h
  \end{array}\]

- Putting it together: 99D7h
Another Example

- Task: 76ABh + 236Ch

- Lower order byte
  ABh +6Ch
  ----
  117h

- Higher order byte
  (add the carry)
  1h +76h +23h
  ----
  9Ah

- Putting it together: 9AD7h
More Instructions

• Add with Carry
  – when addition results in a carry, store this in a flag
  – new add instruction that includes carry if flag set

• Subtract with Borrow
  – when subtraction results in a carry, store this in a flag
  – new subtract instruction that includes carry if flag set
Circuit

Halt
Ctrl
Unit
Instr
CI
LD
CO
Sub

ALU
B
S
A
CO
CI
SUB
CTRL
UNIT
INSTR
STR
CO
HALT
LD
D
>CLK
Q
CI
SUB

Carry
Flag
Each operation is encoded by a byte value.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Code (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>10h</td>
</tr>
<tr>
<td>Store</td>
<td>11h</td>
</tr>
<tr>
<td>Add</td>
<td>20h</td>
</tr>
<tr>
<td>Subtract</td>
<td>21h</td>
</tr>
<tr>
<td>Add with carry</td>
<td>22h</td>
</tr>
<tr>
<td>Subtract with borrow</td>
<td>23h</td>
</tr>
<tr>
<td>Halt</td>
<td>FFh</td>
</tr>
</tbody>
</table>
## Code

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Data</th>
<th>Carry</th>
<th>Accumulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000h</td>
<td>10h</td>
<td>ABh</td>
<td>0</td>
<td>00h</td>
</tr>
<tr>
<td>0001h</td>
<td>20h</td>
<td>6Ch</td>
<td>1</td>
<td>17h</td>
</tr>
<tr>
<td>0002h</td>
<td>11h</td>
<td></td>
<td>1</td>
<td>17h</td>
</tr>
<tr>
<td>0003h</td>
<td>10h</td>
<td>76h</td>
<td>1</td>
<td>76h</td>
</tr>
<tr>
<td>0004h</td>
<td>22h</td>
<td>23h</td>
<td>0</td>
<td>9Ah</td>
</tr>
<tr>
<td>0005h</td>
<td>11h</td>
<td></td>
<td>0</td>
<td>9Ah</td>
</tr>
<tr>
<td>0006h</td>
<td>FFh</td>
<td></td>
<td>0</td>
<td>9Ah</td>
</tr>
</tbody>
</table>
addressing memory
Motivation

- Currently using two memories
  - code memory for instructions
  - data memory

- Very limiting

- Instead:
  - store code and data in same memory
  - add explicit addresses to instructions
Adapted 16-Bit Adder

- Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000h</td>
<td>ABh</td>
</tr>
<tr>
<td>4001h</td>
<td>76h</td>
</tr>
<tr>
<td>4002h</td>
<td>6Ch</td>
</tr>
<tr>
<td>4003h</td>
<td>23h</td>
</tr>
</tbody>
</table>

- Code

<table>
<thead>
<tr>
<th>Address</th>
<th>Bytes</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000h</td>
<td>10h 00h 40h</td>
<td>Load 4000h</td>
</tr>
<tr>
<td>0003h</td>
<td>20h 02h 40h</td>
<td>Add 4002h</td>
</tr>
<tr>
<td>0006h</td>
<td>11h 04h 40h</td>
<td>Store 4004h</td>
</tr>
<tr>
<td>0009h</td>
<td>10h 01h 40h</td>
<td>Load 4001h</td>
</tr>
<tr>
<td>000Ch</td>
<td>22h 03h 40h</td>
<td>Add with carry 4003h</td>
</tr>
<tr>
<td>000Fh</td>
<td>11h 05h 40h</td>
<td>Store 4005h</td>
</tr>
<tr>
<td>0012h</td>
<td>FFh</td>
<td>Halt</td>
</tr>
</tbody>
</table>

Note: Instructions take up 1 or 3 bytes.
Instruction Fetch

3 registers: code and 2 byte data
Instruction Fetch

Transfer bytes from memory to instruction code and data register
Instruction fetch logic determines which register is written to
Instruction Fetch

This is informed by instruction code
Once all registers are filled, execute instruction
Data Paths

• Data needs to be transferred in various ways

• Address passed on to memory (overriding program counter)

• Add/subtract: read byte from memory, pass to ALU

• Load: read byte from memory, store in accumulator

• Store: read byte from accumulator, store in memory

• No detailed wiring worked out here...
multiplication
Plan

- Multiplication by repeated addition
- Pseudo-code

```
load number1 into accumulator
loop
  subtract 1 from number2
  last if number2 = 0
  add number1 to accumulator
store accumulator in result
```
Needed

• Jump
  – set the ripple counter to specified value

• Zero flag
  – detect that subtraction resulted in number 0
  – implemented as flag of the ALU

• Jump if zero
  – check if zero flag is set
  – only then update ripple counter
  – otherwise, do nothing
Zero Flag

- Flag when the ALU operation results in 0
## Instructions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(hex)</td>
</tr>
<tr>
<td>Load</td>
<td>10h</td>
</tr>
<tr>
<td>Store</td>
<td>11h</td>
</tr>
<tr>
<td>Add</td>
<td>20h</td>
</tr>
<tr>
<td>Subtract</td>
<td>21h</td>
</tr>
<tr>
<td>Add with carry</td>
<td>22h</td>
</tr>
<tr>
<td>Subtract with borrow</td>
<td>23h</td>
</tr>
<tr>
<td>Jump</td>
<td>30h</td>
</tr>
<tr>
<td>Jump if zero</td>
<td>31h</td>
</tr>
<tr>
<td>Jump if carry</td>
<td>32h</td>
</tr>
<tr>
<td>Jump if not zero</td>
<td>33h</td>
</tr>
<tr>
<td>Jump if not carry</td>
<td>34h</td>
</tr>
<tr>
<td>Halt</td>
<td>FFh</td>
</tr>
</tbody>
</table>
**Code (8 Bit Version)**

- **Memory**

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000h</td>
<td>0Bh</td>
</tr>
<tr>
<td>4001h</td>
<td>0Fh</td>
</tr>
<tr>
<td>4002h</td>
<td>01h</td>
</tr>
</tbody>
</table>

- **Code**

<table>
<thead>
<tr>
<th>Address</th>
<th>Bytes</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000h</td>
<td>10h 00h 40h</td>
<td>Load 4000h ; load number1</td>
</tr>
<tr>
<td>0003h</td>
<td>11h 03h 40h</td>
<td>Store 4003h ; save in result</td>
</tr>
<tr>
<td>0006h</td>
<td>10h 01h 40h</td>
<td>Load 4001h ; load number2</td>
</tr>
<tr>
<td>0009h</td>
<td>21h 02h 40h</td>
<td>Subtract 4002h ; subtract 1</td>
</tr>
<tr>
<td>000Bh</td>
<td>31h 18h 00h</td>
<td>jump if zero to 0018h ; jump to end if done</td>
</tr>
<tr>
<td>000Fh</td>
<td>10h 03h 40h</td>
<td>load 4003h ; load result</td>
</tr>
<tr>
<td>0012h</td>
<td>20h 00h 40h</td>
<td>add 4000h ; add number1</td>
</tr>
<tr>
<td>0015h</td>
<td>30h 03h 00h</td>
<td>jump to 0003h ; loop</td>
</tr>
<tr>
<td>0018h</td>
<td>FFh</td>
<td>halt ; quit</td>
</tr>
</tbody>
</table>