SCRAM Introduction

Philipp Koehn

13 September 2019
A Simple Computer

• Fully work through a computer
  – circuit
  – assembly code

• Simple but Complete Random Access Machine (SCRAM)
  – every instruction is 8 bit
  – 4 bit for op-code: 9 different operations (of 16 possible)
  – 4 bit for address: 16 bytes of memory

• Background reading on web page: "The SCRAM"
## Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLT</td>
<td>0000</td>
<td>Halt, stop execution</td>
</tr>
<tr>
<td>LDA</td>
<td>0001</td>
<td>Load value from memory into accumulator</td>
</tr>
<tr>
<td>LDI</td>
<td>0010</td>
<td>Indirectly load value from memory into accumulator</td>
</tr>
<tr>
<td>STA</td>
<td>0011</td>
<td>Store value from accumulator into memory</td>
</tr>
<tr>
<td>STI</td>
<td>0100</td>
<td>Indirectly store value from accumulator into memory</td>
</tr>
<tr>
<td>ADD</td>
<td>0101</td>
<td>Add value from memory to accumulator</td>
</tr>
<tr>
<td>SUB</td>
<td>0110</td>
<td>Subtract value from memory from accumulator</td>
</tr>
<tr>
<td>JMP</td>
<td>0111</td>
<td>Jump to specified address</td>
</tr>
<tr>
<td>JPZ</td>
<td>1000</td>
<td>Jump to specified address if zero flag is set</td>
</tr>
</tbody>
</table>
Indirect Load?

- LDA \( x \)
  - loads the value from address \( x \)

- LDI \( x \)
  - looks up the value at address \( x \)
  - treats that value as an address
  - loads the value at that address

- Indirect load = use of pointer variable
### Instruction Encoding

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>LDA</td>
<td>5</td>
</tr>
</tbody>
</table>

Load contents from address 5 into memory
# A Simple Program

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Meaning</th>
<th>data</th>
<th>Operation</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0001</td>
<td>LDA</td>
<td>0100</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0101</td>
<td>ADD</td>
<td>0101</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>0011</td>
<td>STA</td>
<td>0100</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>0111</td>
<td>JMP</td>
<td>0000</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1111</td>
<td>DAT</td>
<td>0000</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1111</td>
<td>DAT</td>
<td>0001</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

- **Note:** DAT is not a real instruction
- **Produce sequence of numbers:**
  
  \[0, 1, 2, 3, 4, \ldots, 255, 0, \ldots\]
components
Memory

- 4 bits to address memory

⇒ 16 different values

⇒ 16 byte address space

- We need to build circuitry to retrieve and store values
Registers

• Accumulator (AC)
  – can be directly accessed from logic units
  – used to store the results of computations

• Program counter (PC)
  – memory address of current instruction

• Instruction register (IR)
  – contains current instruction
  – breaks it down into operation code

• Memory registers
  – memory access register (MAR): address to retrieve value
  – memory buffer register (MBR): retrieved value
Arithmetic Logic Unit (ALU)

- Can do addition and subtraction

- Operands
  - operand 1: accumulator
  - operand 2: address specified in instruction
  - result: accumulator

- Zero flag: result of operation is zero

- Carry flag: operation results in overflow / underflow
Program Counter

- 4 bit current memory address of instruction
- Typically increased by 1 during each instruction execution
- Can also be changed by jump instructions (JMP, JPZ)
Control Logic Unit

- Decodes the op code
- Selects instruction logic
  - Instruction logic: microprogram
    (sequence of register transfers)
- More detail on that in a bit...
### Putting it All Together

<table>
<thead>
<tr>
<th>Program Counter (PC)</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Register (IR)</td>
<td>Memory access register (MAR)</td>
</tr>
<tr>
<td>Control Logic Unit (CLU)</td>
<td>Memory buffer register (MBR)</td>
</tr>
<tr>
<td></td>
<td>Accumulator (A)</td>
</tr>
<tr>
<td></td>
<td>Arithmetric Logic Unit (ALU)</td>
</tr>
</tbody>
</table>
memory
16x8 Bit RAM

- 16 byte of memory
- Inputs
  - address (A)
  - data in (DI)
  - write flag (W)
- Output
  - data out (DO)
Memory with its Registers

- Memory address register (MAR)
- Memory buffer register (MBR)
- Each of them has a write flag (there will be a few more of them...)

16x8 RAM

MAR

A

W

DI

DO

MBR
instruction fetch
Program Counter

- Program counter contains address of current instruction
- This address needs to be passed to MAR
- Program counter also needs an easy way to be incremented
• Content of memory (MBR) is transferred to instruction register
Instruction Fetch

• Program counter (PC) contains address of current instruction

• Step 1: copy PC value to memory access register (MAR)

• Step 2: retrieve address value into memory buffer register (MBR)

• Step 3: copy MBR value into instruction register (IR)

• Step 4: increase PC

• These copy instructions are triggered by write flags
• Copy PC value to memory access register (MAR)
• Retrieve address value into memory buffer register (MBR)
• Copy MBR value into instruction register (IR)
• Increase PC
Micro Program

- We can write these steps in a register transfer language

<table>
<thead>
<tr>
<th>Time</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>MAR $\leftarrow$ PC</td>
</tr>
<tr>
<td>$t_1$</td>
<td>MBR $\leftarrow$ M</td>
</tr>
<tr>
<td>$t_2$</td>
<td>IR $\leftarrow$ MBR</td>
</tr>
<tr>
<td>$t_3$</td>
<td>PC $\leftarrow$ PC + 1</td>
</tr>
</tbody>
</table>

- Execution in time steps $t_n$ triggered by the clock
Parallel Execution

- Increase of the program counter is independent from retrieving data from memory

⇒ These steps can be parallelized

- New micro program

<table>
<thead>
<tr>
<th>Time</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>MAR ← PC</td>
</tr>
<tr>
<td>$t_1$</td>
<td>MBR ← M, PC ← PC + 1</td>
</tr>
<tr>
<td>$t_2$</td>
<td>IR ← MBR</td>
</tr>
</tbody>
</table>
Parallel execution of memory retrieve and program counter increase
control logic unit
Control Logic Unit

Instruction: op-code and data

Control logic unit receives operation code

Op-code is decoded into 9 different operations
Time step needs to be increased or cleared to 0 after each micro command.
Control Logic Unit

Time step triggers one command line in control logic unit