SCRAM Instructions II

Philipp Koehn

23 February 2018
Reminder

• 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 Random Access Machine
  – The SCRAM
Circuit (At This Point)
Instruction Fetch

- Retrieve instruction from memory
- Increase program counter

<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>
Micro Program for STA

- Store value from accumulator

<table>
<thead>
<tr>
<th>Op Code</th>
<th>Time</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₃</td>
<td>t₃</td>
<td>MAR ← IR(D)</td>
</tr>
<tr>
<td>q₃</td>
<td>t₄</td>
<td>M ← AC</td>
</tr>
</tbody>
</table>
$q_3 \; t_3:\quad \text{MAR} \leftarrow \text{IR}(D)$
\( q_3 \ t_4: \ M \leftarrow AC \)
arithmetic logic unit
Arithmetic Logic Unit

- Adds two numbers: $S = A + B$
- With subtraction flag: $S = A - B$
- Overflow handling with carry in (CI) and carry out (CO)
- Zero flag: set if result of operation is 0
• Store result of ALU operation in accumulator (AC)
AC = AC ± B

- Accumulator feeds back into ALU
- Operations are AC = AC + B or AC = AC - B
add
ADD: Add to Accumulator

- Add value from memory address to accumulator

- Steps
  - load value of specified memory address
  - use that value as a memory address (second lookup)
  - store value from second lookup into accumulator
**Micro Program for ADD**

- Load indirectly into accumulator

<table>
<thead>
<tr>
<th>Op Code</th>
<th>Time</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>q&lt;sub&gt;5&lt;/sub&gt;</td>
<td>t&lt;sub&gt;3&lt;/sub&gt;</td>
<td>MAR ← IR(D)</td>
</tr>
<tr>
<td>q&lt;sub&gt;5&lt;/sub&gt;</td>
<td>t&lt;sub&gt;4&lt;/sub&gt;</td>
<td>MBR ← M</td>
</tr>
<tr>
<td>q&lt;sub&gt;5&lt;/sub&gt;</td>
<td>t&lt;sub&gt;5&lt;/sub&gt;</td>
<td>AC ← AC + MBR</td>
</tr>
</tbody>
</table>
\[ q_5 \ t_3 : \ MAR \leftarrow IR(D) \]
$q_5 \ t_4: \ MBR \leftarrow M$
q_5 \ t_5: \ AC \leftarrow AC + MBR
sub
**SUB: Subtract from Accumulator**

- Subtract from accumulator the value from memory

- Same as ADD, just set subtraction flag of ALU
Micro Program for SUB

- Load indirectly into accumulator

<table>
<thead>
<tr>
<th>Op Code</th>
<th>Time</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_5</td>
<td>t_3</td>
<td>MAR ← IR(D)</td>
</tr>
<tr>
<td>q_5</td>
<td>t_4</td>
<td>MBR ← M</td>
</tr>
<tr>
<td>q_5</td>
<td>t_5</td>
<td>AC ← AC - MBR</td>
</tr>
</tbody>
</table>
$q_5 \ t_3: \ MAR \leftarrow IR(D)$
$q_5 \ t_5: \ AC \leftarrow AC + MBR$
jmp
**Program Counter (PC)**

- Position of the next instruction is stored in program counter
- This gets updated during instruction fetch

<table>
<thead>
<tr>
<th>Time</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>$\text{MAR} \leftarrow \text{PC}$</td>
</tr>
<tr>
<td>$t_1$</td>
<td>$\text{MBR} \leftarrow \text{M}$</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$\text{IR} \leftarrow \text{MBR}$</td>
</tr>
<tr>
<td>$\Rightarrow$</td>
<td>$t_3$ $\text{PC} \leftarrow \text{PC} + 1$</td>
</tr>
</tbody>
</table>
JMP: Jump

- Assign value to position of the next instruction

- Sequencing of micro program
  - instruction fetch (includes program counter inc)
  - command-specific micro instructions

- No problem that program counter gets modified twice
Micro Program for JMP

- Change program counter to specified address

<table>
<thead>
<tr>
<th>Op Code</th>
<th>Time</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₇</td>
<td>t₃</td>
<td>PC ← IR(D)</td>
</tr>
</tbody>
</table>
q₇ t₃: PC ← IR(D)
jpz
• Zero flag
  – set when result of a ALU operation is 0
  – stored in flag
Z Flag in Circuit

[Diagram of a computer system showing various components and signals, such as PC, MAR, MBR, 16x8 RAM, Decoder, Control Logic Unit, ALU, and other related components and flags like Z, W, INC, and CLEAR.]
Micro Program for JPZ

- Z flag is a condition for executing a micro program (same as JMP)

\[
\begin{array}{ccccc}
\text{Zero} & \text{Op Code} & \text{Time} & \text{Command} \\
1 & q_7 & t_3 & \text{PC} \leftarrow \text{IR(D)} \\
\end{array}
\]

- If not set, no micro program is executed