Appendix B
The design of VS0: a very simple CPU
Instruction set

• Just 4 instructions

LOAD M - Copy into Accumulator the value from memory at address M
STORE M - Save Accumulator value into memory at address M
ADD M - Sum values of Accumulator and of memory at address M and put the result into the Accumulator
JUMP A - Execute in the next step the instruction stored at address A of memory
Registers and Memory

- The bare minimum
  PC   - Program Counter
  IR   - Instruction Register
  MAR  - Memory Address Register
  MBR  - Memory Buffer Register
  AC   - Accumulator
- All registers have 8 bits
- 64 ($2^6$) bytes of memory, each with 8 bits
**Instruction format**

- 2 bits for the opcode
- 6 bits for the address (\(b_5\) is the MSB, \(b_0\) is the LSB)

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>0 0 b_5 b_4 b_3 b_2 b_1 b_0</td>
</tr>
<tr>
<td>STORE</td>
<td>1 1 b_5 b_4 b_3 b_2 b_1 b_0</td>
</tr>
<tr>
<td>ADD</td>
<td>0 1 b_5 b_4 b_3 b_2 b_1 b_0</td>
</tr>
<tr>
<td>JUMP</td>
<td>1 0 b_5 b_4 b_3 b_2 b_1 b_0</td>
</tr>
</tbody>
</table>

- No interrupt mechanism
ALU’s organization

- Only capable of adding (signal CA) two 8 bits number with a possible carry-in (signal CC)
- No overflow signal
- One addend is the Accumulator
- The other addend is the selection between PC and a memory address (through C6 and C14)
- ALU’s output is stored into an internal buffer register
ALU’s internal structure
Internal schema

- Data Bus
  - C5 → MBR
  - C12
  - PC
  - C13 → IR
  - C4
  - C16
  - C10
  - C11
  - C15

- Address Bus
  - C0 → MAR
  - C2 → C14
  - C10
  - C7
  - C9

- Clocks for mOPs
  - t1
  - t2
  - t3
  - t4

- Control Unit
  - CC
  - CR
  - CW
  - CA

- All Cn, n=0..16, but for n=1,3,8
Micro-operations (1)

• Fetch
  t1:   MAR <- PC       C2
  t2:   MBR <- memory; PC+1;
        C0  C5  C14  CA  CC  CR
  t3:   PC <- ALU; IR <- MBR
        C4  C15

• Execute ADD
  t1:   MAR <- IR_{addr}   C16
  t2:   MBR <- memory      C0  C5  CR
  t3:   AC+MBR             C6  C7  CA
  t4:   AC <- ALU          C9
Micro-operations (2)

- Execute LOAD
  
  \[
  \begin{align*}
  t1: & \quad \text{MAR} \leftarrow \text{IR}_{\text{addr}} \quad \text{C16} \\
  t2: & \quad \text{MBR} \leftarrow \text{memory} \quad \text{C0} \quad \text{C5} \quad \text{CR} \\
  t3: & \quad \text{AC} \leftarrow \text{MBR} \quad \text{C10}
  \end{align*}
  \]

- Execute STORE
  
  \[
  \begin{align*}
  t1: & \quad \text{MAR} \leftarrow \text{IR}_{\text{addr}}; \quad \text{MBR} \leftarrow \text{AC} \quad \text{C11} \quad \text{C16} \\
  t2: & \quad \text{memory} \leftarrow \text{MBR} \quad \text{C0} \quad \text{C12} \quad \text{CW}
  \end{align*}
  \]

- Execute JUMP
  
  \[
  \begin{align*}
  t1: & \quad \text{PC} \leftarrow \text{IR}_{\text{addr}} \quad \text{C13}
  \end{align*}
  \]
Decoding instructions

- Inside the Control Unit a 2-to-4 decoder provides L, S, A, J signals denoting which instruction is currently in IR.
Micro-operations (3)

- Generate \( t_1, t_2, t_3, t_4 \) from the clock through a base-4 counter and a 2-to-4 decoder
- Distinguish between Fetch and Execute with a 1-bit state register (can be inside the Control Unit)
- For each control signal \( C_n \) write the boolean expression for its activation in terms of status (Fetch/Execute), mOP step being executed \( (t_1, t_2, t_3, t_4) \), and operation to be executed \( (L, S, A, J) \), by scanning the list of activated control signals for each step of each mOP
Generating clocks for mOPs

• Counter can be reset at the last step of each mOP to optimize performances
• Reset signal: $Ft_3 + F't_1J + F't_2S + F't_3L$
• N.B.: $t_1=00$, $t_2=01$, $t_3=10$, $t_4=11$  !!!
State representation

- Control unit can be in the state of fetch (F=1) or in the state of execute (F=0)
- Status changes are activated during the last mOP step of each phase of fetch or execute

\[ F_{n+1} = F_n t_3' + F_n t_4 A + F_n t_3 L + F_n t_2 S + F_n t_1 J \]
## Activation of control signals

<table>
<thead>
<tr>
<th>Control Signal</th>
<th>Boolean expression</th>
<th>Control Signal</th>
<th>Boolean expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>$F_t^2 + F_t'^2A + F_t'^2L + F_t'^2S$</td>
<td>C12</td>
<td>$F_t'^2S$</td>
</tr>
<tr>
<td>C2</td>
<td>$F_t^1$</td>
<td>C13</td>
<td>$F_t'^1J$</td>
</tr>
<tr>
<td>C4</td>
<td>$F_t^3$</td>
<td>C14</td>
<td>$F_t^2$</td>
</tr>
<tr>
<td>C5</td>
<td>$F_t^2 + F_t'^2A + F_t'^2L$</td>
<td>C15</td>
<td>$F_t^3$</td>
</tr>
<tr>
<td>C6</td>
<td>$F_t'^3A$</td>
<td>C16</td>
<td>$F_t'^1A + F_t'^1L + F_t'^1S$</td>
</tr>
<tr>
<td>C7</td>
<td>$F_t'^3A$</td>
<td>CC</td>
<td>$F_t^2$</td>
</tr>
<tr>
<td>C9</td>
<td>$F_t'^4A$</td>
<td>CA</td>
<td>$F_t^2 + F_t'^3A$</td>
</tr>
<tr>
<td>C10</td>
<td>$F_t^3L$</td>
<td>CR</td>
<td>$F_t^2 + F_t'^2A + F_t'^2L$</td>
</tr>
<tr>
<td>C11</td>
<td>$F_t^3 + F_t'^1S$</td>
<td>CW</td>
<td>$F_t'^2S$</td>
</tr>
</tbody>
</table>
A note on boolean expressions (1)

- Boolean expressions written have been derived directly from inspection of mOPs.
- The theory of circuit synthesis tells us to examine what happens in general to each output signal for each possible combination of input signals \((t_1, t_2, t_3, t_4, L, S, A, J)\) and state signal \((F)\).
- Writing, e.g., \(F' t_2 L\) could be wrong, since the exact and complete term is \(F' t_1' t_2 t_3' t_4' LS' A' J'\) : this is not equivalent to the former, which corresponds to \(F'(t_1+t_1')t_2(t_3+t_3')(t_4+t_4')L(S+S')(A+A')(J+J')\).
A note on boolean expressions (2)

- But we know that among \( t_1, t_2, t_3, \) and \( t_4 \) only and exactly one can be true, therefore we can substitute, e.g., \( t_1' t_2 \) with \( (t_1+t_1')t_2 \) knowing that the condition \( t_1 t_2 \) can never be true and hence derive the correct simpler term \( t_2 \) (in other words, \( t_1 t_2 \) is a don’t care condition).
- For signals L, S, A, and J, if one of them is true then all the others are false and the same reasoning above applies.
- Finally, there are also those situations (e.g., for C2 activation in mOP \( t_1 \) during the fetch phase) where we don’t care at all about which of signals L,S,A,J is true.
## Global optimization of signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Boolean expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC, C14</td>
<td>$F_t_2$</td>
</tr>
<tr>
<td>C2</td>
<td>$F_t_1$</td>
</tr>
<tr>
<td>C4, C15</td>
<td>$F_t_3$</td>
</tr>
<tr>
<td>C5, CR</td>
<td>$CC + F't_2A + F't_2L$</td>
</tr>
<tr>
<td>C6, C7</td>
<td>$F't_3A$</td>
</tr>
<tr>
<td>C9</td>
<td>$F't_4A$</td>
</tr>
<tr>
<td>C10</td>
<td>$F't_3L$</td>
</tr>
<tr>
<td>C11</td>
<td>$C4 + F't_1S$</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signal</th>
<th>Boolean expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>C13</td>
<td>$F't_1J$</td>
</tr>
<tr>
<td>CW, C12</td>
<td>$F't_2S$</td>
</tr>
<tr>
<td>C16</td>
<td>$F't_1A + F't_1L + F't_1S$</td>
</tr>
<tr>
<td>C0</td>
<td>$C5 + CW$</td>
</tr>
<tr>
<td>CA</td>
<td>$CC + C6$</td>
</tr>
<tr>
<td>$F_{n+1}$</td>
<td>$F_t_3' + C9 + C10 + CW + C13$</td>
</tr>
</tbody>
</table>
Additional considerations

- Do we need both state signal and instruction signals L,S,A,J to activate control signals?
  - e.g. in the activation expression for C7, instead of $F' t_3 A$, can we just write $t_3 A$?
    - no, because if the previously fetched instruction was also an ADD then C7 is (wrongly) activated also during mOP step $t_3$ in the fetch phase
    - hence we need both state signal and instruction signals
- Do we need an explicit representation for state?
  - no, if we use for the execution phases a different set of clock signals $t_4, t_5, t_6, t_7$
  - What changes using this approach? What do we lose?
No Explicit State: Micro-operations (1)

- **Fetch**
  
  t1: \( \text{MAR} \leftarrow \text{PC} \)  
  
  t2: \( \text{MBR} \leftarrow \text{memory}; \text{PC}+1; \)  
      \[
      \begin{array}{cccccc}
      \text{C0} & \text{C5} & \text{C14} & \text{CA} & \text{CC} & \text{CR}
      \end{array}
      \]
  
  t3: \( \text{PC} \leftarrow \text{ALU}; \text{IR} \leftarrow \text{MBR} \)  
      \[
      \begin{array}{cccc}
      \text{C4} & \text{C15}
      \end{array}
      \]

- **Execute ADD**
  
  t4: \( \text{MAR} \leftarrow \text{IR}_{\text{addr}} \)  
      \[
      \begin{array}{c}
      \text{C16}
      \end{array}
      \]
  
  t5: \( \text{MBR} \leftarrow \text{memory} \)  
      \[
      \begin{array}{ccc}
      \text{C0} & \text{C5} & \text{CR}
      \end{array}
      \]
  
  t6: \( \text{AC}+\text{MBR} \)  
      \[
      \begin{array}{ccc}
      \text{C6} & \text{C7} & \text{CA}
      \end{array}
      \]
  
  t7: \( \text{AC} \leftarrow \text{ALU} \)  
      \[
      \begin{array}{c}
      \text{C9}
      \end{array}
      \]
No Explicit State: Micro-operations (2)

• Execute LOAD
  t4: \( MAR \leftarrow IR_{\text{addr}} \) \quad C16
  t5: \( MBR \leftarrow \text{memory} \) \quad C0 \quad C5 \quad CR
  t6: \( AC \leftarrow MBR \) \quad C10

• Execute STORE
  t4: \( MAR \leftarrow IR_{\text{addr}}; \quad MBR \leftarrow AC \) \quad C11 \quad C16
  t5: \( \text{memory} \leftarrow MBR \) \quad C0 \quad C12 \quad CW

• Execute JUMP
  t4: \( PC \leftarrow IR_{\text{addr}} \) \quad C13
No Explicit State: Micro-operations (3)

- Generate $t_1, t_2, t_3, t_4, t_5, t_6, t_7$ from the clock through a base-8 counter and a 3-to-8 decoder (possibly use a counter with reset for optimization).

- For each control signal $C_n$ write the boolean expression for its activation in terms of mOP step being executed ($t_1, t_2, t_3, t_4, t_5, t_6, t_7$), and operation to be executed (L,S,A,J), by scanning the list of activated control signals for each step of each mOP.
# No Explicit State: Activation of control signals

<table>
<thead>
<tr>
<th>Control Signal</th>
<th>Boolean expression</th>
<th>Control Signal</th>
<th>Boolean expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>$t_2 + t_5 A + t_5 L + t_5 S$</td>
<td>C12</td>
<td>$t_5 S$</td>
</tr>
<tr>
<td>C2</td>
<td>$t_1$</td>
<td>C13</td>
<td>$t_4 J$</td>
</tr>
<tr>
<td>C4</td>
<td>$t_3$</td>
<td>C14</td>
<td>$t_2$</td>
</tr>
<tr>
<td>C5</td>
<td>$t_2 + t_5 A + t_5 L$</td>
<td>C15</td>
<td>$t_3$</td>
</tr>
<tr>
<td>C6</td>
<td>$t_6 A$</td>
<td>C16</td>
<td>$t_4 A + t_4 L + t_4 S$</td>
</tr>
<tr>
<td>C7</td>
<td>$t_6 A$</td>
<td>CC</td>
<td>$t_2$</td>
</tr>
<tr>
<td>C9</td>
<td>$t_7 A$</td>
<td>CA</td>
<td>$t_2 + t_6 A$</td>
</tr>
<tr>
<td>C10</td>
<td>$t_3 L$</td>
<td>CR</td>
<td>$t_2 + t_5 A + t_5 L$</td>
</tr>
<tr>
<td>C11</td>
<td>$t_3 + t_4 S$</td>
<td>CW</td>
<td>$t_5 S$</td>
</tr>
</tbody>
</table>
The complete circuit

• All circuital elements (including the Control Unit) have now been defined and it is known how to realize them
• Try drawing the complete circuit for the CPU and the memory!!
• It is a long but worthwhile task
• Do it in hierarchical stages: first layout modules and afterwards layout gates within modules
• In the real life they use CAD systems for electronic circuit design!
A trivial program

• Give at location SUM the sum of four numbers stored in locations of memory N1, N2, N3, N4

; Location SUM is distinct from N1, N2, N3, N4
LOAD N1 ; AC <- N1
ADD N2 ; AC <- N1+N2
ADD N3 ; AC <- N1+N2+N3
ADD N4 ; AC <- N1+N2+N3+N4
STORE SUM ; SUM <- N1+N2+N3+N4
Control Unit’s implementation with micro-programmed control

- For the implementation of CU with a micro-programmed approach we **do not need**:
  - signals \( t_1 \) ... \( t_n \) marking different mOPs
  - state register distinguishing between fetch and execute
- Even the IR decoder is not really needed, but we may use it depending on the CW structure
- Structure of CW and structure of Sequencing Logic are strictly related: a CW with more information needs a simpler Sequencing Logic and vice-versa
CW and sequencing mOPs

- CW has two address fields (SmA and JmA) of 5 bits each
  - SmA is the next CW address in case of sequential execution
  - JmA is the next CW address in case of jump
  - Fields are empty when the choice is forced
- A 2-way multiplexer is used to select between SmA and JmA and hence choose the next CW to be executed
- Selection line (SEL) for multiplexer is activated by a circuit in the Sequential Logic whose structure depends on the structure of jump conditions in CW
  - No jump flags
  - One jump flag (K) only for end-of-mOP
  - Jump flags both for end-of-mOP and for selecting the proper micro-procedure during the CPU execution phase
Generic structure of CU

CONTROL UNIT

Control Memory

Control Signals | Jump Conditions | Seq. CW Addr | Jump CW Addr

Sequencing Logic

MUX

IR

CAR

Rev. 1.4.1 (2010-11) by Enrico Nardelli
CW without jump conditions: CU’s structure
CW without jump conditions: Sequencing Logic

- If there are no flags in CW the selection between SmA and JmA may use only the state of CU, represented by CAR value.
- Towards the end of CU execution cycle, CAR contains the address of current CW in execution hence the value of such an address is used to drive the selection of next CW.
- A CAR decoder provides $I_n$ signals telling that CW at address $n$ is being executed.
- A 2-to-4 decoder on the two most significant bits of IR is needed to understand which CPU instruction is being executed and to provide L, S, A, and J signals.
- Signal for selection line (0 to select SmA, 1 for JmA) is:
  - $SEL = I_3 + I_6 + I_{12} + I_{16} + I_{17} + I_7L + I_8S + I_9A + I_{10}J$
- Both decoders are part of the Sequencing Logic.
## CW without jump conditions: Control Memory

<table>
<thead>
<tr>
<th>Micro Procedure</th>
<th>mA</th>
<th>C0</th>
<th>C2</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
<th>C9</th>
<th>C10</th>
<th>C11</th>
<th>C12</th>
<th>C13</th>
<th>C14</th>
<th>C15</th>
<th>C16</th>
<th>CC</th>
<th>CA</th>
<th>CR</th>
<th>CW</th>
<th>S</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Store</td>
<td>11</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jump</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rev. 1.4.1 (2010-11) by Enrico Nardelli
CW with one jump condition: CU’s structure
CW with one jump condition: Sequencing Logic

- A jump flag (K) is used to mark the last mOP of each micro-procedure (but for the Execute one)
- $I_n$ signals provided by the CAR decoder now are only needed during the Execute micro-procedure
- A 2-to-4 decoder on the two most significant bits of IR is needed to understand which CPU instruction is being executed and to provide L, S, A, and J signals
- Signal for selection line (0 to select SmA, 1 for JmA) is
  - $SEL = K + I_7L + I_8S + I_9A + I_{10}J$
- Sequencing Logic is independent from the location of any micro-procedure in Control Memory, but for the Execute one
# CW with one jump condition: Control Memory

<table>
<thead>
<tr>
<th>Micro Procedure</th>
<th>mA</th>
<th>C0</th>
<th>C2</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
<th>C9</th>
<th>C10</th>
<th>C11</th>
<th>C12</th>
<th>C13</th>
<th>C14</th>
<th>C15</th>
<th>C16</th>
<th>CC</th>
<th>CA</th>
<th>CR</th>
<th>CW</th>
<th>K</th>
<th>S mA</th>
<th>J mA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>13</td>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
<td>17</td>
</tr>
<tr>
<td>Store</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jump</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Rev. 1.4.1 (2010-11) by Enrico Nardelli
CW with many jump conditions:
CU’s structure
**CW with many jump conditions: Sequencing logic**

- A jump flag (K) is used to mark the last mOP of each micro-procedure.
- Four jump flags (E_L, E_S, E_A, E_J) mark the four mOPs in the Execute micro-procedure.
- There is no need now for a CAR decoder: this is obtained at the cost of a longer CW.
- A 2-to-4 decoder on the two most significant bits of IR is needed to understand which CPU instruction is being executed and to provide L, S, A, and J signals.
- Signal for selection line (0 to select SmA, 1 for JmA) is
  - \(\text{SEL} = K + E_L L + E_S S + E_A A + E_J J\)
- Sequencing Logic is now fully independent from the location of any micro-procedure in Control Memory.
### CW with many jump conditions: Control Memory

| Micro Procedure | mA | C0 | C2 | C4 | C5 | C6 | C7 | C9 | C10 | C11 | C12 | C13 | C14 | C15 | C16 | CC | CA | CR | CW | EL | ES | EA | EJ | K | S mA | J mA |
|-----------------|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|----|----|
| Fetch           | 1  | 1  |    |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
|                 | 2  | 1  | 1  |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
|                 | 3  | 1  |    |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
| Load            | 4  |    |    |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
|                 | 5  | 1  | 1  |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
|                 | 6  |    |    |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
| Execute         | 7  |    |    |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
|                 | 8  |    |    |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
|                 | 9  |    |    |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
|                 | 10 |    |    |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
| Store           | 11 |    |    |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
|                 | 12 | 1  | 1  |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
| Add             | 13 |    |    |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
|                 | 14 | 1  | 1  |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
|                 | 15 |    | 1  | 1  |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
|                 | 16 |    |    | 1  |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |
| Jump            | 17 |    |    |    |    |    |    |    |     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |     |     |

Rev. 1.4.1 (2010-11) by Enrico Nardelli
An internal schema with single bus
**ALU changes**

- ALU needs a buffer (with reset) also for input
Micro-operations (1) Single Bus

- Fetch
  - one more step
  
  t1: MAR <- PC  C2  C13
  
  t2: MBR <- memory  C16  C15  CR
      PC+1  C2  C7  CT  CA  CC
  
  t3: PC <- ALU  C8  C1
  
  t4: IR <- MBR  C10  C3
Micro-operations (2) Single Bus

- Execute ADD
  - reorganization of micro-operations

  t1:  \( \text{MAR} \leftarrow \text{Ir}_{\text{addr}} \quad C4 \quad C13 \)
  t2:  \( \text{MBR} \leftarrow \text{memory} \quad C16 \quad C15 \quad \text{CR} \)
        \( \text{ALU} \leftarrow \text{AC} \quad C6 \quad C9 \)
  t3:  \( \text{MBR+ALU} \quad C10 \quad C7 \quad \text{CA} \)
  t4:  \( \text{AC} \leftarrow \text{ALU} \quad C8 \quad C5 \)
Micro-operations (3) Single Bus

- Execute LOAD

  t1: \[ \text{MAR} \leftarrow \text{IR}_{\text{addr}} \quad C4 \quad C13 \]
  t2: \[ \text{MBR} \leftarrow \text{memory} \quad C16 \quad C15 \quad CR \]
  t3: \[ \text{AC} \leftarrow \text{MBR} \quad C10 \quad C5 \]
Micro-operations (4) Single Bus

• Execute STORE
  ▪ one more step
  t1: \( \text{MAR} \leftarrow \text{IR}_{\text{addr}} \)  \( \text{C4} \text{ C13} \)
  t2: \( \text{MBR} \leftarrow \text{AC} \)  \( \text{C6} \text{ C11} \)
  t3: memory \( \leftarrow \text{MBR} \)  \( \text{C14} \text{ C16} \text{ CW} \)

• Execute JUMP
  t1: \( \text{PC} \leftarrow \text{IR}_{\text{addr}} \)  \( \text{C14} \text{ C2} \)
Completion of single bus

• Continue development as shown before
• Decide whether to explicitly represent state or not
• Decide whether to implement a hardwired CU or a micro-programmed one
• In the latter case, decide the structure of the control word
Other simple design variations

- **Try them** (even together) to understand consequences of various design decisions!
  1. Add to the ALU the capability to provide Zero or Overflow signal and use a JUMP conditional to the signal value instead of the unconditional JUMP
  2. Use an internal CPU schema with two internal buses to connect CPU elements instead of direct paths
  3. Use two variants of ADD. One, specified by $b_5=0$, having as parameter the address of memory cell, written in the byte right after the one with ADD. The other, specified by $b_5=1$, having as argument the number to be added written in bits $b_4-b_0$
  4. Use a micro-programmed CU with just one address field
  5. Study if it is possible to avoid the use of the 2-to-4 IR decoder by means of a different organization of the micro-procedure for the CPU execution phase