### 5.3 Microprogramming

- Control store and microbranching
- Horizontal and vertical microprogramming

---

#### Microprogramming: Basic Idea

**• Recall control sequence for 1-bus SRC**

<table>
<thead>
<tr>
<th>Step</th>
<th>Concrete RTN</th>
<th>Control Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>MA ← PC: C ← PC + 4;</td>
<td>(PC_{out}, MA_{in}, INC4, C_{in}, ) Read</td>
</tr>
<tr>
<td>T1</td>
<td>MD ← M[MA]: PC ← C;</td>
<td>(C_{out}, PC_{in}, ) Wait</td>
</tr>
<tr>
<td>T2</td>
<td>IR ← MD;</td>
<td>(MD_{out}, IR_{in} )</td>
</tr>
<tr>
<td>T3</td>
<td>A ← R[rb];</td>
<td>Grb, (R_{out}, A_{in} )</td>
</tr>
<tr>
<td>T4</td>
<td>C ← A + R[rc];</td>
<td>Grc, (R_{out}, ADD, C_{in} )</td>
</tr>
<tr>
<td>T5</td>
<td>R[ra] ← C;</td>
<td>(C_{out}, Gra, R_{in}, ) End</td>
</tr>
</tbody>
</table>

Control unit job is to generate the sequence of control signals
How about building a computer to do this?
The Microcode Engine

- A computer to generate control signals is much simpler than an ordinary computer
- At the simplest, it just reads the control signals in order from a read-only memory
- The memory is called the control store
- A control store word, or microinstruction, contains a bit pattern telling which control signals are true in a specific step
- The major issue is determining the order in which microinstructions are read

Fig 5.16 Block Diagram of Microcoded Control Unit

Microinstruction has branch control, branch address, and control signal fields
Microprogram counter can be set from several sources to do the required sequencing
Parts of the Microprogrammed Control Unit

- Since the control signals are just read from memory, the main function is sequencing.
- This is reflected in the several ways the μPC can be loaded:
  - Output of incrementer—μPC + 1
  - PLA output—start address for a macroinstruction
  - Branch address from μinstruction
  - External source—say for exception or reset
- Micro conditional branches can depend on condition codes, data path state, external signals, etc.

Contents of a Microinstruction

- Main component is list of 1/0 control signal values.
- There is a branch address in the control store.
- There are branch control bits to determine when to use the branch address and when to use μPC + 1.
Fig 5.17 The Control Store

<table>
<thead>
<tr>
<th>Microaddress</th>
<th>( \mu )Code for instruction fetch</th>
<th>( \mu )Code for add</th>
<th>( \mu )Code for br</th>
<th>( \mu )Code for shr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 2^n-1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( m \) bits wide
- \( k \) \( \mu \)branch control bits
- \( c \) control signals
- \( n \) branch addr. bits

Common instruction fetch sequence
Separate sequences for each (macro) instruction
Wide words

Tbl 5.2 Control Signals for the add Instruction

<table>
<thead>
<tr>
<th>Address</th>
<th>End</th>
<th>POut</th>
<th>Cout</th>
<th>Cout</th>
<th>Rout</th>
<th>Cout</th>
<th>Cin</th>
<th>Cin</th>
<th>Rin</th>
<th>Rin</th>
<th>Rin4</th>
<th>Rin</th>
<th>Rin4</th>
<th>ADD</th>
<th>ADD</th>
<th>ADD</th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>***</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>102</td>
<td>***</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>103</td>
<td>***</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>***</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>201</td>
<td>***</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>202</td>
<td>***</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Addresses 101–103 are the instruction fetch
- Addresses 200–202 do the add
- Change of \( \mu \)control from 103 to 200 uses a kind of \( \mu \)branch
Uses for \( \mu \)branching in the Microprogrammed Control Unit

- (1) Branch to start of \( \mu \)code for a specific inst.
- (2) Conditional control signals, e.g. \( \text{CON} \rightarrow \text{PC}_{\text{in}} \)
- (3) Looping on conditions, e.g. \( n \neq 0 \rightarrow \ldots \text{Goto6} \)
- Conditions will control \( \mu \)branches instead of being ANDed with control signals
- Microbranches are frequent and control store addresses are short, so it is reasonable to have a \( \mu \)branch address field in every \( \mu \) instruction

Illustration of \( \mu \)branching Control Logic

- We illustrate a \( \mu \)branching control scheme by a machine having condition code bits N and Z
- Branch control has 2 parts:
  - (1) selecting the input applied to the \( \mu \)PC and
  - (2) specifying whether this input or \( \mu \)PC + 1 is used
- We allow 4 possible inputs to \( \mu \)PC
  - The incremented value \( \mu \)PC + 1
  - The PLA lookup table for the start of a macroinstruction
  - An externally supplied address
  - The branch address field in the \( \mu \)instruction word
Fig 5.18 Branching Controls in the Microcoded Control Unit

5 branch conditions
- NotN
- N
- NotZ
- Z
- Unconditional

To 1 of 4 places
- Next µinstruction
- PLA
- External address
- Branch address

Some Possible µbranches Using the Illustrated Logic (Refer to Tbl 5.3)

<table>
<thead>
<tr>
<th>Mux Sel</th>
<th>NotN</th>
<th>N</th>
<th>NotZ</th>
<th>Z</th>
<th>Control Signals</th>
<th>Branch Address</th>
<th>Branching action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>XXX</td>
<td>None—next instruction</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>XXX</td>
<td>Branch to output of PLA</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>XXX</td>
<td>Br if Z to External Addr.</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>300</td>
<td>Br if N to 300 (else next)</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>206</td>
<td>Br if N to 206 (else next)</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>204</td>
<td>Br to 204</td>
</tr>
</tbody>
</table>

If the control signals are all zero, the µinstruction only does a test
Otherwise test is combined with data path activity
Horizontal versus Vertical Microcode Schemes

In horizontal microcode, each control signal is represented by a bit in the \( \mu \) instruction.

In vertical microcode, a set of true control signals is represented by a shorter code. The name horizontal implies fewer control store words of more bits per word.

Vertical \( \mu \) code only allows RTs in a step for which there is a vertical \( \mu \) instruction code.

Thus vertical \( \mu \) code may take more control store words of fewer bits.

---

**Fig 5.20** Completely Horizontal and Vertical Microcoding
Saving Control Store Bits with Horizontal Microcode

Some control signals cannot possibly be true at the same time:

- One and only one ALU function can be selected
- Only one register out gate can be true with a single bus
- Memory read and write cannot be true at the same step

A set of m such signals can be encoded using \( \log_2 m \) bits (\( \log_2 (m + 1) \) to allow for no signal true)

The raw control signals can then be generated by a \( k \) to \( 2^k \) decoder, where \( 2^k \geq m \) (or \( 2^k \geq m + 1 \))

This is a compromise between horizontal and vertical encoding.

Fig 5.19  A Somewhat Vertical Encoding

<table>
<thead>
<tr>
<th>( \mu IR )</th>
<th>F5</th>
<th>( \ldots )</th>
<th>F8</th>
</tr>
</thead>
</table>

4 \( \rightarrow \) 16 decoder

16 ALU control signals

3 \( \rightarrow \) 8 decoder

7 Reg\(_{out}\) control signals

Scheme would save \((16 + 7) - (4 + 3) = 16\) bits/word in the case illustrated.
A Microprogrammed Control Unit for the 1-Bus SRC

- Using the 1-bus SRC data path design gives a specific set of control signals
- There are no condition codes, but data path signals CON and \( n = 0 \) will need to be tested
- We will use \( \mu \)branches \( BrCON, Brn = 0 \), and \( Brn \neq 0 \)
- We adopt the clocking logic of Fig. 4.14
- Logic for exception and reset signals is added to the microcode sequencer logic
- Exception and reset are assumed to have been synchronized to the clock

### Tbl 5.4 The add Instruction

<table>
<thead>
<tr>
<th>Addr.</th>
<th>Mux Clr</th>
<th>Br( \mathit{Un} )</th>
<th>Br( \mathit{CON} )</th>
<th>Br( n=0 )</th>
<th>End</th>
<th>PC( \mathit{Out} )</th>
<th>MA( n )</th>
<th>Other Control Signals</th>
<th>Br Addr.</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>00</td>
<td>0 0 0 0 0 0</td>
<td>1 1</td>
<td></td>
<td></td>
<td>XXX</td>
<td>XXX</td>
<td>XXX</td>
<td></td>
<td>MA &lt;- PC; C &lt;- PC+4;</td>
</tr>
<tr>
<td>101</td>
<td>00</td>
<td>0 0 0 0 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td>XXX</td>
<td>XXX</td>
<td>XXX</td>
<td></td>
<td>MD &lt;- M[MA]; PC &lt;- C;</td>
</tr>
<tr>
<td>102</td>
<td>01</td>
<td>1 0 0 0 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td>XXX</td>
<td>XXX</td>
<td>XXX</td>
<td></td>
<td>IR &lt;- MD; ( \mu )PC &lt;- PLA;</td>
</tr>
<tr>
<td>200</td>
<td>00</td>
<td>0 0 0 0 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td>XXX</td>
<td>XXX</td>
<td>XXX</td>
<td></td>
<td>A &lt;- R[rb];</td>
</tr>
<tr>
<td>201</td>
<td>00</td>
<td>0 0 0 0 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td>XXX</td>
<td>XXX</td>
<td>XXX</td>
<td></td>
<td>C &lt;- A + R[rc];</td>
</tr>
<tr>
<td>202</td>
<td>11</td>
<td>1 0 0 0 0 1</td>
<td>0 0</td>
<td>0 0</td>
<td>100</td>
<td>000</td>
<td>100</td>
<td>XXX</td>
<td></td>
<td>R[ra] &lt;- C; ( \mu )PC &lt;- 100;</td>
</tr>
</tbody>
</table>

Microbranching to the output of the PLA is shown at 102
Microbranch to 100 at 202 starts next fetch
**Fig 5.16 Block Diagram of Microcoded Control Unit**

- Microinstruction has branch control, branch address, and control signal fields.
- Microprogram counter can be set from several sources to do the required sequencing.

![Block Diagram of Microcoded Control Unit](image)

**Fig 5.21 SRC Microcode Sequencer**

![SRC Microcode Sequencer Diagram](image)
Other Microprogramming Issues

- A hardware push-down stack for the μPC can turn repeated μsequences into μsubroutines
- Vertical μcode can be implemented using a horizontal μengine, sometimes called nanocode
Microprogramming Summary

- Microprogramming is a design method with a target of easing the design task and allowing for easy design change or multiple compatible implementations of the same instruction set.