Microprogramming Main Points/Terminology

- Difference between hardwired control unit and microprogrammed control unit
- Microprogram
  - Microinstruction
  - Macroinstruction
- Control store and micro-branching
- Horizontal and vertical microprogramming

Important Trends/Concepts that Led to Microprogramming

- 1940's – Stored program computers
  - Program instructions can be stored in memory along with data and can be manipulated like data
- 1947 – MIT Whirlwind
  - Real-time flight simulator
  - Control store
    - Two-dimensional lattice
      - Rows – time sequence
      - Columns – control signals

MIT Whirlwind Control Store

Historical Background (cont’d)

- 1951 – Maurice Wilkes
  - Sequencing of control signals within the computer was similar to the sequencing actions required in a regular program.
  - A stored program to represent the sequences of control signals. Called it microprogramming
  - Divide machine instructions into subinstructions (microinstructions) that implement the instruction set of the machine
  - Full set of microinstructions made up the microprogram
1970s

- Complex instruction sets
  - Trend towards instruction sets very similar to high-level languages

Important Trends that Led to Microprogramming

- **Control logic design**
  - Complex and error-prone
  - Need a simpler method of developing the control logic for a computer
  - Writing a program
    - Simpler than designing the logic
    - Easier to change

- Benefits microprogramming offers
  - More complex instructions can be implemented
  - More primitive than assembly
  - Reduces field changes to defects

Is Microprogramming Still Used Today?

- ROM’s used to be faster than RAM, but not any longer
- Control stores were implemented on ROM
- Instruction sets have become much simpler (than the 1970s)
  - Less complexity
- Computer-aided design tools have improved

**Yes, but benefits need to be weighed against costs:**
- Complex ISAs such as IA-32 sometimes have more complex instructions implemented as microprograms.
  - Pentium, Pentium 4, special-purpose processors, etc.

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>C_PCA, MA, Inc4, C, Read</td>
</tr>
<tr>
<td>T1.</td>
<td>MD ← M[MA]; PC ← C;</td>
<td>C/url, PC, Wait</td>
</tr>
<tr>
<td>T2.</td>
<td>IR ← MD;</td>
<td>MD/url, IR</td>
</tr>
<tr>
<td>T3.</td>
<td>A ← R[rb];</td>
<td>Grb, R/url, A</td>
</tr>
<tr>
<td>T4.</td>
<td>C ← A + R[rc];</td>
<td>Grc, R/url, ADD, C</td>
</tr>
<tr>
<td>T5.</td>
<td>R[ra] ← C;</td>
<td>C/url, Gra, R/url, End</td>
</tr>
</tbody>
</table>

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

Microprogramming Process

1. Develop microinstruction set
2. Write microprogram
3. Microassemble microprogram
4. Place the microassembled program (microcode) onto PLA or ROM

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
  - Separate from program memory
- A control store word, or microinstruction, contains a bit pattern telling which control signals are true in a specific step
- The major issue is sequencing
  - What order to read instructions?
What Goes in a Microinstruction?
Designing the Microinstruction set

- How many fields?
  - What does each field represent?
  - What control signals are controlled by each field?
  - How will sequencing (next microinstruction to execute) be determined?

Fig 5.22  Block Diagram of a Microcoded Control Unit

- Microinstruction has branch control, branch address, and control signal fields
- Micro-program counter can be set from several sources to do the required sequencing
  - Next sequential
  - Start address of next macro inst
  - Microinst specified address

Parts of the Microprogrammed Control Unit

- Control signals are just read from memory, the main function is sequencing
- This is reflected in the several ways the μPC can be loaded
  1. Output of incrementer—μPC+1
  2. PLA output—start address for a macroinstruction
  3. Branch address from μinstruction
  4. 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

Microinstruction format

- 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

Figure 5.23: Layout of the Control Store

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

Size and Shape of System RAM vs Control Store

- System RAM is one byte wide × $2^{32}$ bytes deep
- Assume control store has 128 instructions, 128 bits wide, with 8 steps each
- Control store would be 16 bytes wide, but only 128x8 or 1024 words deep
Table 5.2: Microinstruction Control Signals for the add Instruction

<table>
<thead>
<tr>
<th>Address</th>
<th>Branch Control</th>
<th>Start Code</th>
<th>Branch Addr</th>
<th>Control</th>
<th>Cntrl</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>101 ***</td>
<td>1 0 0 0 0 0 0 0</td>
<td>1 0 0 0 1 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>1 1 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>102 ***</td>
<td>0 1 0 0 0 0 0 0</td>
<td>0 1 0 0 1 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>103 ***</td>
<td>0 0 1 0 0 0 0 0</td>
<td>0 0 1 0 1 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>200 ***</td>
<td>0 0 0 1 0 0 0 0</td>
<td>0 0 0 1 1 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>201 ***</td>
<td>0 0 0 0 1 0 0 0</td>
<td>0 0 0 0 0 1 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>202 ***</td>
<td>0 1 0 0 0 0 0 0</td>
<td>0 1 0 0 0 1 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Addresses 101–103 are the instruction fetch
- Addresses 200–202 do the add
- Change of µcontrol from 103 to 200 uses a kind of µbranch

Table 5.2 Continued...

<table>
<thead>
<tr>
<th>Address</th>
<th>Branch Control</th>
<th>Start Code</th>
<th>Branch Addr</th>
<th>Control</th>
<th>Cntrl</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>103 ***</td>
<td>0 0 1 0 0 0 0 0</td>
<td>0 0 1 0 1 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>200 ***</td>
<td>0 0 0 1 0 0 0 0</td>
<td>0 0 0 1 1 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>201 ***</td>
<td>0 0 0 0 1 0 0 0</td>
<td>0 0 0 0 0 1 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>202 ***</td>
<td>0 1 0 0 0 0 0 0</td>
<td>0 1 0 0 0 1 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0</td>
</tr>
</tbody>
</table>

Uses for µbranching in the Microprogrammed Control Unit

1) Branch to start of µcode for a specific inst.
2) Conditional control signals, e.g. CON → PC
3) Looping on conditions, e.g. n ≠ 0 → Goto6
Conditions will control µbranches instead of being ANDed with control signals
Microbranches are frequent and control store addresses are short, so it is reasonable to have a µbranch address field in every µinstruction

Fig 5.24 Branching Controls in the Microcoded Control Unit

Fig 5.25 A Somewhat Vertical Encoding

Horizontal Versus Vertical Microcode Schemes

- In **horizontal** microcode, each control signal is represented by a bit in the µinstruction
  - Fewer control store words of more bits per word
- In **vertical** microcode, a set of true control signals is represented by a shorter code
  - Vertical µcode only allows RTs in a step for which there is a vertical µinstruction code
  - Thus vertical µcode may take more control store words of fewer bits

Some Possible µbranches Using the Illustrated Logic

- If the control signals are all zero, the µinst. only does a test
- Otherwise test is combined with data path activity
Figure 5.26 Completely Horizontal and Vertical Microcoding

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 µbranches BrCON, Brn=0, & Brn≠0.
- We adopt the clocking logic of Fig. 4.3 on p. 4-20.
- 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.

Table 5.4 Microinstructions for SRC add

<table>
<thead>
<tr>
<th>Address</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Other Control Signals</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</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>XXX</td>
<td>XXX</td>
</tr>
<tr>
<td>101</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>XXX</td>
<td>XXX</td>
</tr>
<tr>
<td>102</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>XXX</td>
<td>XXX</td>
</tr>
<tr>
<td>200</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>XXX</td>
<td>XXX</td>
</tr>
<tr>
<td>201</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>XXX</td>
<td>XXX</td>
</tr>
<tr>
<td>202</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>100</td>
<td>102</td>
</tr>
</tbody>
</table>

- Microbranching to the output of the PLA is shown at 102.
- Microbranch to 100 at 202 starts next fetch.

Getting the PLA Output in Time for the Microbranch

- So that the input to the PLA is correct for the µbranch in 102, it has to come from MD, not IR.
- An alternative is to use see-thru latches for IR so the op code can pass through IR to PLA before the end of the clock cycle.

See-thru Latch Hardware for IR So µPC Can Load Immediately

- Data must have time to get from MD across Bus, through IR, through the PLA, and satisfy µPC set up time before trailing edge of S.

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 log2m bits (log2(m+1) to allow for no signal true).
- The raw control signals can then be generated by a k to 2k decoder, where 2k ≥ m (or 2k ≥ m+1).
- This is a compromise between horizontal and vertical encoding.

Saving Control Store Bits With Horizontal Microcode

Microbranching to the output of the PLA is shown at 102.

Microbranch to 100 at 202 starts next fetch.
### Other Microprogramming Issues

- Multi-way branches: often an instruction can have 4-8 cases, say address modes
  - Could take 2-3 successive \( \mu \) branches, i.e. clock pulses
  - The bits selecting the case can be ORed into the branch address of the \( \mu \) instruction to get a several way branch
  - Say if 2 bits were ORed into the 3rd & 4th bits from the low end, 4 possible addresses ending in 0000, 0100, 1000, and 1100 would be generated as branch targets
  - Advantage is a multi-way branch in one clock
- A hardware push-down stack for the \( \mu \) PC can turn repeated \( \mu \) sequences into \( \mu \) subroutines
- Vertical \( \mu \) code can be implemented using a horizontal engine, sometimes called nanocode