University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Microprogrammed Control Unit Design

LC-3 State Transition Diagram Has Few Outgoing Arcs

Take another look at Patt and Patel's LC-3 state transition diagram (Figure C.2).

Does it remind you of anything?

Like a flow chart, each state (except for decode) has only one or two arcs leaving it.

Microprogrammed Control Treats States as Instructions

Can we treat a state diagram as a program?

Each state has specific RTL
- expressed as **control words**
- (a set of control signals for a state),
- which we can think of as **microinstructions**.

Let's **store the microinstructions in a ROM**,
- and **use the state ID as an address**
- **to read the microinstruction** for that state.

This approach is called **microprogrammed control unit design**.

Let's Assume 5-Bit State IDs for LC-3

Ignoring interrupts and privilege, and
- including the extension mentioned earlier* to handle **JSR(R)** logic with **PCMUX** (one extra control signal, so 26 total),
- we **need fewer than 32 states** for the **LC-3 FSM**.

So **state IDs require only 5 bits**.

*Without changing the datapath or keeping an old **JSRR** bug, the **FSM** requires 33 states.

Microprogrammed Control Allows a Smaller ROM

The control ROM is thus **2^5×26-bit**.

Each cycle, the microprogrammed control unit
- applies the 5-bit state ID to the control ROM address, and
- uses the 26 bits read from the control ROM
- to drive the datapath.

Notice that **IR** is not used as part of the address, so the control ROM is **much smaller than that needed for hardwired control**.

Microsequencing Manages the Order of Microinstructions

But how do we handle transitions between states?

That problem is called **microsequencing** (or sometimes just sequencing).

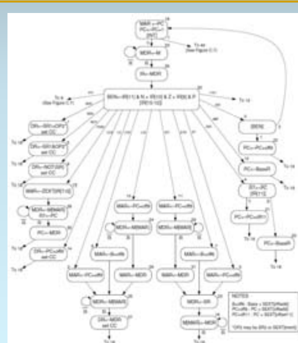
Let's look again at the state transition diagram.

Almost All States Have Only One or Two Outgoing Arcs

Most states have only a single arc.

Some states (such as memory access) have two arcs.

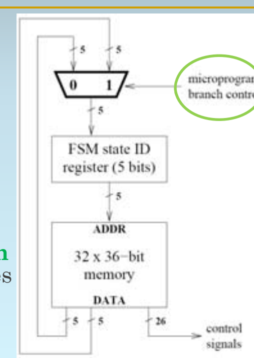
Let's **add two state IDs to each microinstruction**.



Basic Organization for Microprogrammed Control

A microinstruction thus consists of
- 26 bits of control signals,
- one 5-bit next state ID, and
- a second 5-bit next state ID.

A **microprogram branch control signal** determines which next state to use.



Branches Occur for Two Reasons in the LC-3 FSM

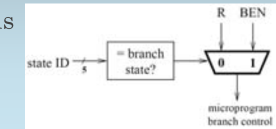
What is the microprogram branch control?

Looking at the state diagram, there are only two reasons* for branches:
- memory ready signal **R**
- branch enable signal **BEN**

*We removed the branch on **IR[11]** for **JSR(R)** with our datapath extension, and we are ignoring interrupts and privilege.

Branch Control Requires a Comparison and a Mux

Simple logic thus suffices for microprogram branch control.



For the branch state, we use **BEN** to decide the next state.

For all others, we use **R**.

When no branch is needed, both next state IDs are the same, so the **R** value doesn't matter.

Choose State Numbers to Simplify Decode Branching

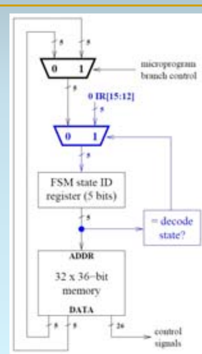
What's left? **Decode**!

We handle decode using a trick from Patt and Patel:
- use states 0 through 15
- as the first execution state for the corresponding opcode.

For example, ADD is state 1.

And AND is state 5.

See the **blue extensions**.



Assign State IDs to Complete the Design

All that's left is to assign state IDs
- in the range 16 to 31
- to the fetch, decode, and later execution states.

Then calculate all the bits and put them into the control ROM.

Note that the control ROM totals 1152 bits, about 30% as many as needed with our smallest hardwired design.

