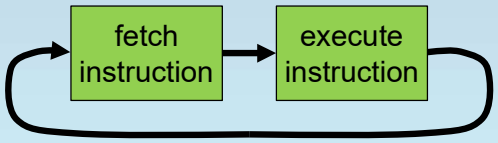
University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Hardwired Control Unit Design

A Processor's Control Unit Processes Instructions

What does the control unit do?

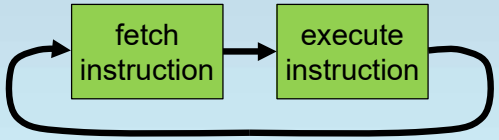


Forever. And ever. And ever.

It's pretty exciting.

One Approach: Use a Counter to Drive the State Sequence

How can we implement a control unit?



What **if** we assume that **each task requires a fixed number of cycles**.

In that case, **we can use a counter**!

Hardwired Control Uses Combinational Logic

The control unit **FSM** then does the following:

- given as inputs
  - the counter value,
  - the **IR** (for execution only), and
  - signals from the datapath,
- generate the datapath control signals (outputs) using combinational logic.

Such an approach is called **hardwired control**.

Single-Cycle Hardwired Control is Rare

For a simple enough **ISA**, a powerful datapath might be able to process an instruction in every cycle.

In such a case, the control unit is said to be a **single-cycle, hardwired control** unit.

But that approach requires a complex datapath, and usually a slow clock.

Consider Multi-Cycle Hardwired Control for LC-3

Let's use Patt and Patel's **LC-3** datapath and state transition diagram as an example.

That datapath can neither fetch nor execute an instruction in a single cycle.

But we can still **use combinational logic**.

In this case, the control unit design approach is called **multi-cycle, hardwired control**.

A 3-Bit Binary Counter Suffices for an LC-3 Design

How many cycles do we need to process an instruction?

Fetch requires **three states**.

The longest instruction **execution** sequence is **five states** (**LDI** and **STI**).

So the total is **eight states**.

We can **use a 3-bit binary counter**.

Counter Values for LC-3 Multi-Cycle Hardwired Control

counter value	meaning
0	first fetch state
1	second fetch state
2	third fetch state
3	first execute state
4	second execute state
5	third execute state
6	fourth execute state
7	fifth execute state

What Happened to Decode?

Patt and Patel use a different control unit design strategy that requires an explicit state for decode. We'll discuss that strategy later.

A multi-cycle, hardwired control unit does not require a decode state.

Instead, the **combinational logic uses the IR bits directly as inputs**, and **IR** has the instruction bits after fetch completes.

What About the Shorter Instructions?

Executing an **ADD** only takes one cycle.

But we allocated five cycles for instruction execution to allow for **LDI** and **STI**.

Do we need to wait for four cycles after an **ADD** to start the next instruction?

No! Add a RESET signal to the counter.

Resetting the counter starts a new fetch.

And **RESET** is just another control signal.

What About Memory Accesses?

Memory access may be slow!

Does the clock speed need to be **slow enough to allow memory to complete an access**?

No! Add a PAUSE signal to the counter.

When reading or writing memory, use **PAUSE** until memory finishes.

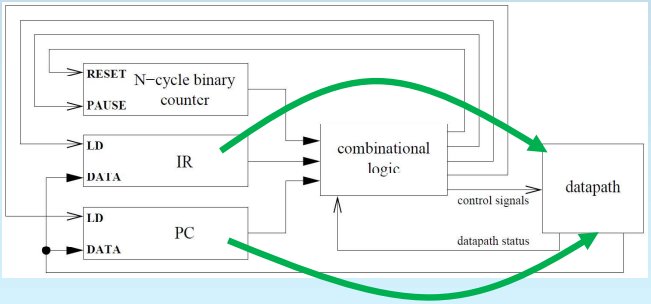
PAUSE is another control signal.

And the **clock can run at the speed of the logic**.

Illustration of Multi-Cycle Hardwired Control

(**N = 8** for the **LC-3** design.)

The datapath may use IR and PC, too.



Combinational Logic Seems Too Complex to Build

How complex is the combinational logic?

For the **LC-3 ISA**, the **PC** does not directly affect control.

So we have as **FSM** inputs:
- 3 bits of counter state,
- 16 bits of **IR**, and
- datapath status signals.

Maybe around 24 bits.

Get your K-map pens ready!

Human Ingenuity to the Rescue!

Here's where human ingenuity comes to the rescue.

Patt and Patel carefully **designed the LC-3 ISA to simplify control**.

You have seen many examples already when we examined the control signals.

So what inputs do we really need for the combinational logic?

Careful Design Reduces the Input Bits to 10

A careful examination of FSM states gives:
- 3 bits of counter state,
- the opcode (**IR[15:12]**),
- **IR[11]** for **JSR(R)** (an instruction for ECE220), and
- two datapath signals:
  - memory ready signal **R**, and
  - branch enable signal **BEN**.

So only 10 bits of input.

We Need to Compute 27 10-Input Functions

How many control signals are there?
- 25 from the P&P datapath
- counter **RESET**
- counter **PAUSE**

27 10-input functions is still a bit challenging with pencil and paper.

Is there an easier way?

Use a Memory to Replace Combinational Logic

Instead of designing the logic, we can **use a read-only memory (ROM)**:

- The **10 bits of input act as an address**.
- When we read from the memory, we need 27 control signals.
- So we need **27-bit addressability**.

In other words, a **2^{10}×27-bit memory**.

That's only 27,648 bits.

Can We Reduce the Size of the Control Memory?

But smaller memories are faster, so let's think a little.

The datapath in Patt and Patel was designed for their control unit.

If we change the datapath slightly, we can reduce the control ROM size.

Let's see how.

Let's Calculate PAUSE in the Datapath

In the state transition diagram,

- the memory ready signal **R**
- is used to stall the **FSM** (self-loops)
- until memory finishes an access.

The current design uses **R** as an **FSM** input.

FSM states that access memory use **R** to generate **PAUSE**.

Let's **move the logic for calculating PAUSE** out of the control **ROM** and **into the datapath**.

Adding PAUSE Logic Cuts the Control ROM Size in Half

First, add a control signal, **WAIT-MEM**, that indicates a need to wait for memory.

FSM states that stall set **WAIT-MEM = 1**.

Then add logic: **PAUSE = R' · WAIT-MEM**.

The number of control signals is still 27 (added **WAIT-MEM** but removed **PAUSE**).

But the number of inputs is now only 9!

So the control ROM size is reduced by half: 2^9×27-bit.

Another Opportunity: Use of BEN to Calculate RESET

The branch enable signal **BEN**
- is used to return to fetch
- when a branch is not taken.

Again, the initial design performs this computation implicitly in the control ROM.

The first **BR** execution state sets counter **RESET = 1** when **BEN = 0**.

All instruction types set counter **RESET = 1** when they have finished execution.

Adding Logic for RESET Compution Cuts ROM in Half

Let's add two new control signals:
- **BR-RESET**: reset the counter to avoid changing the **PC** with an untaken branch
- **INST-DONE**: reset the counter (instruction execution is done).

Then we can add logic to the datapath for the counter's **RESET** input:

RESET = INST-DONE + BEN' · BR-RESET

And memory again shrinks to 2^8×28-bit.

Adding IR[11] Logic to the PCMUX Also Shrinks ROM

Finally, **IR[11]** is used only for **JSR(R)**.

Let's connect **SR1** to **PCMUX**'s fourth input.

Then we can use **IR[11]** directly in the datapath (via another control signal) to control **PCMUX** when appropriate.

The control ROM then shrinks to **2^7×29-bit**, or 3,712 bits total.

Less than one-seventh of the original design!

Some Details on the Last Change

With the extra logic, we can execute **JSR(R)** with a single **FSM** state.

PC crosses the bus to be stored in **R7**.

The address generation adder calculates the address for **JSR**.

The new **PCMUX** input carries the **BaseR** value for **JSRR**.

When the new control signal is high (only in the **JSR(R)** execution state), **IR[11]** is used to select the correct **PCMUX** input.