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

# ECE 120: Introduction to Computing

Instruction Processing

## The Control Unit Executes Instructions on the Datapath

You saw some examples of LC-3 instructions.

The FSM executes those instructions using the LC-3 datapath (through control signals).

But the **datapath can only do so much**!

Consider the memory:
- one **read** in a cycle, **OR**
- one **write** in a cycle, but **NOT both**!
- In fact, memory might take many cycles for one operation (remember the **R** signal)?

## How Can We Fetch and Perform a Load or a Store?

Now think back to the instructions.

The **instructions are in memory**.

We need to **use memory to fetch** each instruction.

You saw **load** and **store** instructions.

Each **requires a memory operation**.

**What should we do?!**

## Break Instruction Processing into Steps

Don't panic!

There's nothing new here.

For a peanut butter sandwich, we open the jar, then get the peanut butter out.

To open a car, we press once to unlock the driver's door, and a second time for the other doors.

We just need to **break instruction processing into more than one step**.

The FSM will **use a separate state for each step**.

## Types of Activities for Processing Instructions

**What kinds of things do we need to do?**

1. FETCH an instruction.
2. DECODE it (look at the opcode).

**always**

3. EVALUATE ADDRESS to calculate the address of any memory access.
4. FETCH OPERANDS from the register file.
5. EXECUTE the operation requested.
6. STORE RESULT back to the register file or to memory.

**some-times**

## Focus on the Steps that are Always Needed

Don't worry too much about the categories.

Each instruction requires a specific set of steps for execution on a datapath.

What steps are required depends on the datapath.

We will look more carefully at the P&P datapath in a few weeks (see Notes Sec. 4.1).

For now, let's **focus on the parts that always happen**: **FETCH** and **DECODE**.

## Fetch Must Copy Bits from Memory to the IR

Let's think about instruction fetch.

**Where are the bits of the instruction?**

**In memory.**

**Where do we want those bits to be?**

**In the IR.**

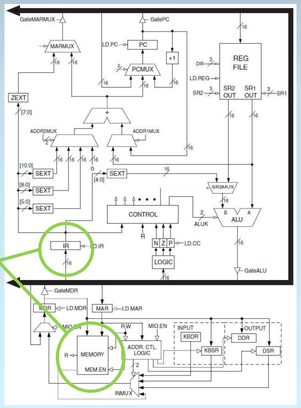


Figure is Patt and Patel Fig. C.3.

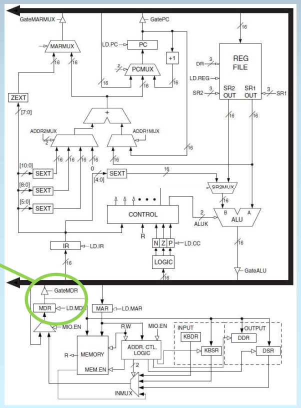
## How Can We Move Bits into the IR? Use the Bus

Now, let's go in reverse.

**What's the last step?**

Memory can't write to the **IR**.

- But memory can write to the **MDR**,
- which can be copied through the bus to the **IR**.



## Building Backwards from Our Goal: Instruction Bits in IR

So the last step in fetch is the following:

**IR ← MDR**

and the previous step fills **MDR**.

In other words, we perform a **read** operation.

**But what is the address for a read?**

Memory must use the **MAR**. Thus…

**MDR ← M[MAR]**

## Building Backwards from Our Goal: Instruction Bits in IR

Here's the end of our **FETCH** sequence:

**state N: MDR ← M[MAR]**

**state N + 1: IR ← MDR**

**How do we set MAR?**

Let's go back to the datapath.

## Copy PC into MAR Across the Bus

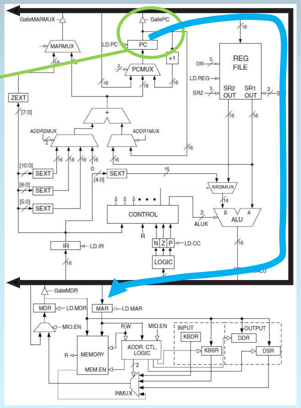
**Where is the next instruction?**

In the **PC**.

So we need to **copy PC to the MAR**.

**How?**

We can do so **across the bus**.

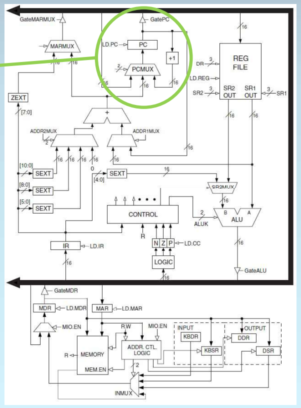


## Increment PC in the Same Cycle

In the same cycle, let's **add 1 to PC**.

Then **PC will point to** the instruction after the one we have fetched (**the new "next" instruction**).

(Note that incrementing PC does not use the bus.)



## Now We Have the Full FETCH Sequence

**state 1: MAR ← PC, PC ← PC + 1**

**state 2: MDR ← M[MAR]**

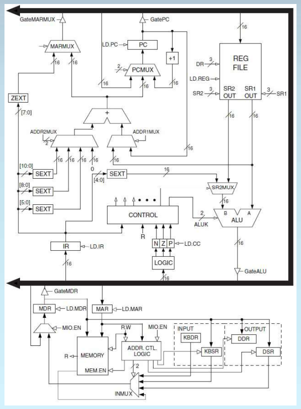
**state 3: IR ← MDR**

Remember that **RTL actions happen in parallel**, so the value of **PC** sent to **MAR** is the value before fetch.

But **when the LC-3 executes an instruction, PC holds that instruction's address PLUS 1**.

## Let's Go Forward and Look at Control Signals

Now, let's go forward and see how each of the three states needed to FETCH an instruction can be accomplished with control signals in the datapath.



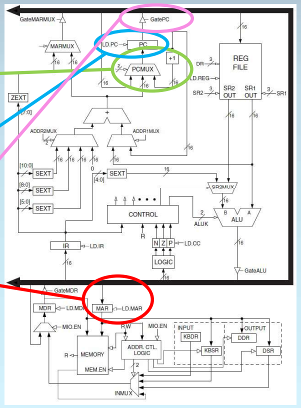
## In the First State of FETCH … (Control Signals)

**PCMUX** is set to select its **PC + 1** input.

And **LD.PC** is set to store **PC+1** back to PC.

**GatePC** is set to copy **PC** on to the bus.

**LD.MAR** is also set, storing **PC** from the bus into **MAR**.



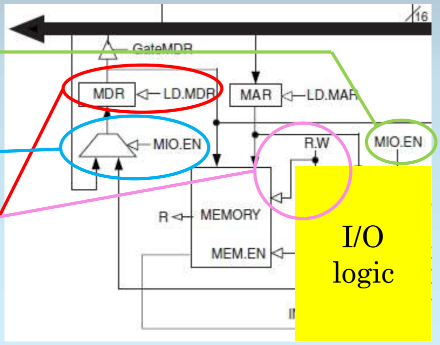
## In the Second State of FETCH … (Control Signals)

The memory is enabled by setting **MIO.EN** to 1.

Notice that **MIO.EN** also controls this mux.

A read operation is requested using **R.W**.

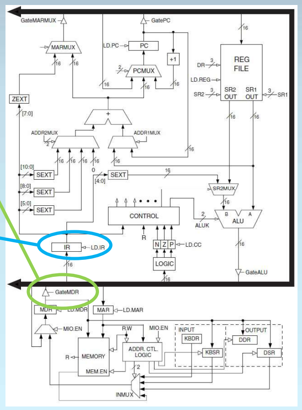
**LD.MDR** is set to store the bits read from memory to the **MDR**.



## In the Third State of FETCH … (Control Signals)

**Gate.MDR** is set to write **MDR** on to the bus.

And **LD.IR** is set to store **MDR** from the bus into the **IR**.



## After FETCH, the FSM Must DECODE the Opcode

Each type of instruction **uses a distinct sequence of FSM states** for execution.

To enter the correct sequence, we **need an FSM transition** (a clock cycle).

But the FSM **cannot decode** the opcode **until the opcode is in IR[15:12]***.

So we have **state #4: DECODE** (transition to an opcode-specific state).

*Bits 15, 14, 13, and 12 of the IR.

## 5+ States for Processing an Instruction on P&P's Datapath

**Instruction processing** on P&P's LC-3 datapath thus **requires the following** for each instruction:

- **state 1: MAR ← PC, PC ← PC + 1**
- **state 2: MDR ← M[MAR]**
- **state 3: IR ← MDR**
- **state 4: DECODE**
- **(variable): execute the instruction**

**FETCH**

## Some States (Accessing Memory) Require Many Cycles

What's the relationship between FSM states and cycles?

**Each state requires at least one cycle.**

However, some states may have self-loops, allowing the FSM to stay in those states indefinitely.

In particular, **memory is slow relative to the processor**.

**Memory access states**, such as the second fetch state, typically **require more than one cycle**.

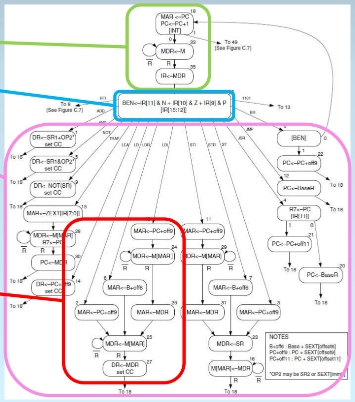
## FSM State Diagram (Patt and Patel Figure C.2)

FETCH (3 states)

DECODE (1 state)

EXECUTE:

- one sequence per opcode
- one to five states in length, with
- some overlap between opcode sequences.



## A Computer Simply Executes Instructions

After finishing any execution sequence, the FSM returns to the first FETCH state.

So the FSM does the following

1. Fetch an instruction.
2. Decode the instruction.
3. Execute the instruction.
4. Go back to Step #1.

Forever.

**That's a computer!**

## A Closing Thought on the FSM

Think back to the start of class.

If I had asked you: **how many bits do you need to control a computer?**

Because I couldn't have asked, "How many bits of state do you need for the (high-level) FSM?

What would you have guessed?

For Patt and Patel's microarchitecture, **the answer is 6 bits**.

I think that's pretty amazing.