Today's Discussion

How do we add support for new instructions (aka processor extensions) to a processor?

2 things to consider:

- Datapath
- Control Logic

Review: What is available in our datapaths?

- Program Counter (PC)
- Memory
    - Separate Instruction and Data memory for Single Cycle
    - Combined (Von Neumann) Instruction + Data memory for Multi Cycle
    - Data memory has read and write capability
- Register File
    - Read and write capability
- Arithmetic and Logic Unit (ALU), extra adders for single cycle
- Muxes, wires, control Logic
- For the Multi Cycle: registers to hold intermediate values

Tricks with the ALU

Remember these tricks from Lab 2:

```
NOR(X, X)      == NOR(X, 0)    == ~X
~(NOR(X, Y))  == ~(~(X | Y))  == X | Y
NOR(~X, ~Y)   == ~(~X | ~Y)   == X & Y
X + (~Y + 1)  == X + (-Y)     == X - Y
X + X                          == X * 2 == X << 1
X * 2^Y                        == X << Y
```

Components used by each Opcode

cheat sheet

Instr.	Read PC, Access Instr. Mem.	Read Reg.	ALU	Data Mem. Access	Write PC	Write Reg.
add	✓	✓	✓			✓
nor	✓	✓	✓			✓
lw	✓	✓	✓	✓ (Read)		✓
sw	✓	✓	✓	✓ (Write)		
beq	✓	✓	✓x2		✓	
jalr	✓	✓			✓	✓
noop	✓					
halt	✓					

Review: Single vs Multi Cycle

For Single Cycle:

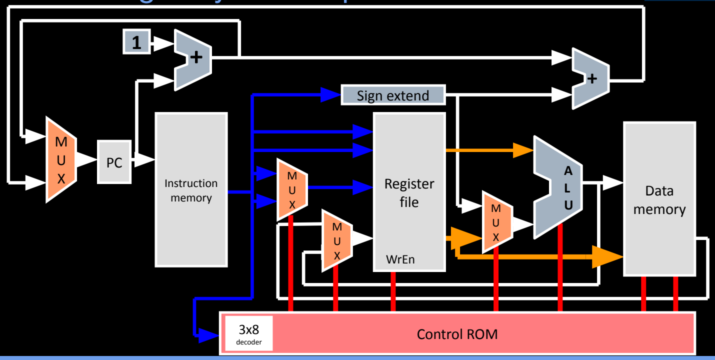
- Simplified logic since there is 1 cycle per instruction
- If we need a component twice, we must **duplicate** it

For Multi Cycle:

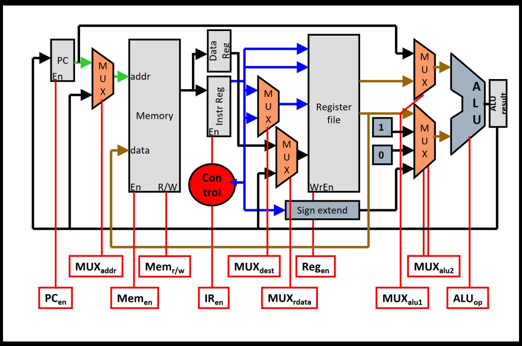
- 2+ cycles per instruction, must define **control logic** per cycle
- If we need a component twice, use it in different cycles
- Attempt to parallelize independent tasks on different components

For both: only 1 value can be on a wire in any cycle. Every wire should only have 1 input. Adding a mux allows multiple inputs

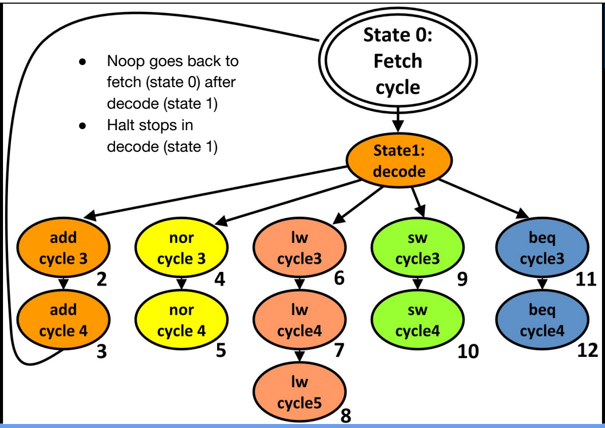
LC2K Single-Cycle Datapath



LC2K Multi-cycle Datapath



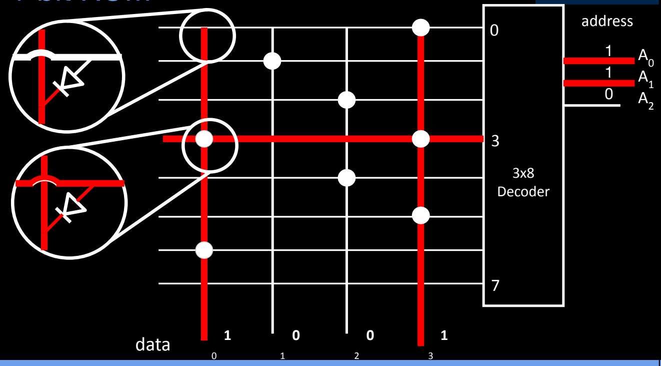
Multi-Cycle State Machine



- Noop goes back to fetch (state 0) after decode (state 1)
- Halt stops in decode (state 1)

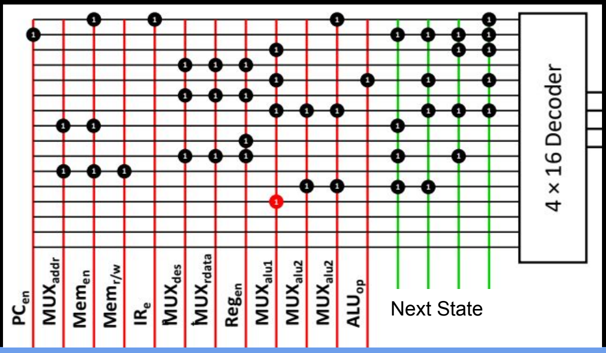
8-entry 4-bit ROM

- A diode only allows current to flow in one direction.
- It prevents a '1' from propagating to other lines.



Use a ROM to Select Inputs in Our MUXes

The ROM specifies which inputs and outputs each component uses for each state



Performance

cheat sheet

IRON LAW: Execution Time = #Instructions*CPI*ClockPeriod = #Cycles*ClockPeriod

Clock Period

- Single Cycle: Latency of Slowest Instruction
- Multi Cycle: Latency of Slowest Cycle (generally slowest datapath component)

#Cycles

- Single Cycle: #Instructions
- Multi Cycle: #Instructions * sum over all opcodes(%opcodes*cycles for opcode)

CPI

- Single Cycle: 1 (it's in the name)
- Multi Cycle: #Cycles/#Instructions

#Cycles Per Opcode (MULTICYCLE ONLY)

- add/nor/sw/beq: 4 cycles
- lw: 5 cycles
- noop/halt: 2 cycles
- jalr: Don't worry about it

Lab Assignment Problem 2

We need to add this instruction to the single cycle datapath:

`[destReg] = memory[ 2 * [regB] ]`

Each operator in the C expression will tell us which components and inputs we need.

For example, we can convert: `2 * [regB]` → `[regB] + [regB]`

Our datapath allows us to add, but we can't add regB to regB!

Need to modify the inputs to fix.

Lab Assignment Problem 2

After modifying the datapath, we can order the steps to execute the instruction: `[destReg] = memory[ [regB] + [regB] ]`

- Fetch Instruction from Memory (ALWAYS!)
- Read registers, i.e. regB
- Add regB to regB (output wire is ALU result)
- Load memory at ALU result
- Write memory result to register, i.e. destReg

Then, you can write the control ROM for these steps.

Lab Assignment Problem 3

We need to add this instruction to the multi cycle datapath:

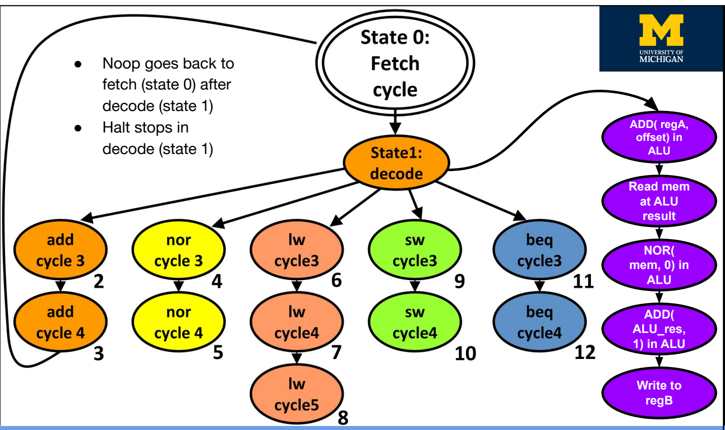
`[regB] = -memory[ [regA] + offset ]`

What's here or missing?

- We can fetch and read registers
- We can add regA to offset
- We can load memory at the ALU result
- We can write to regB from the ALU or memory
- Missing: we need to do ALU ops on memory and intermediate results from the ALU. `-mem == ~(mem | 0) + 1`

Problem 3 Multi Cycle FSM

- Noop goes back to fetch (state 0) after decode (state 1)
- Halt stops in decode (state 1)



On the exam, you will be expected to:

- Parse the instruction
- Define steps & cycles
- Modify the datapath
- Write the control ROM