ECE 385 – Digital Systems Laboratory

Lecture 11 –Experiment 6.2 – SLC-3 CPU & Review

Zuofu Cheng

Spring 2019

Link to Course Website

ECE ILLINOIS

ILLINOIS

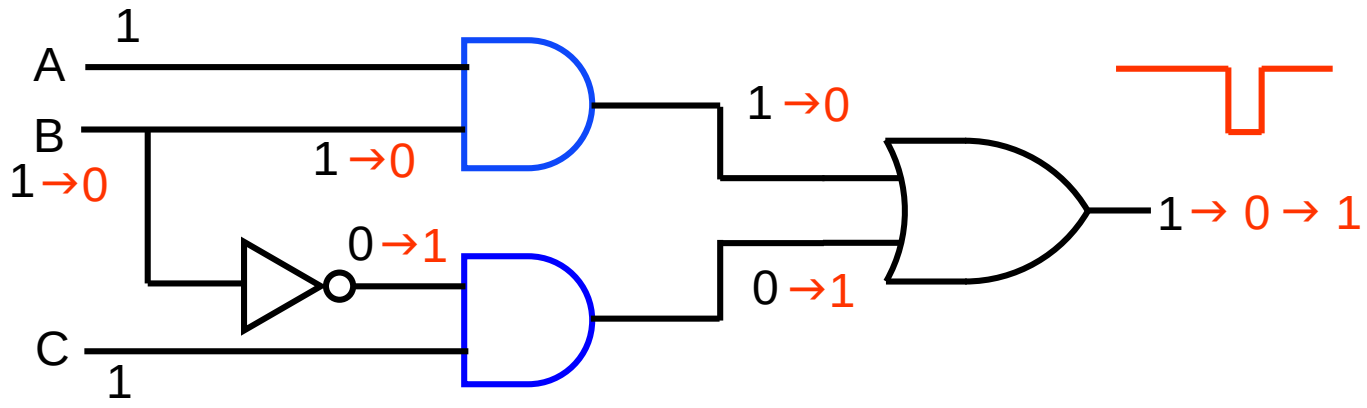
Midterm 1 Logistics

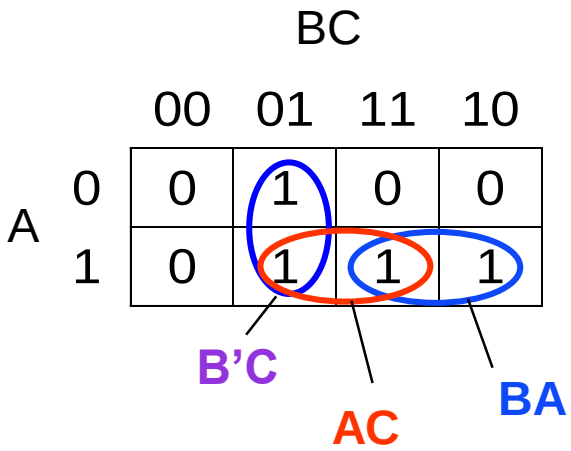
- Midterm Exam I on 2/25 in class (Monday)
- Rooms assignments by last name
  - **1015** ECEB: **A-G**
  - **1002** ECEB: **H-W**
  - **3015** ECEB: **X-Z**
  - **Bring Student ID to Exam and #2 pencil**
- 30 multiple choice questions, closed notes, 40 minutes, no calculator
- Review abstract for labs and block diagrams, review lecture slides & **your own notes** from lecture
- Bulk of questions on labs 1-5 (some on general debugging, lectures)
- HKN review session 1-3 PM Sunday 2/24 (2017 ECEB)

General Studying Strategies

- Make sure you can sketch out the block diagrams for labs 1-5
  - This is the most important point
  - Exam will test to make sure you understand labs as an individual student
  - Also make sure you can write a short (couple sentence) description for each module
- In addition, make sure you understand how the "core algorithm" in each lab works
  - Adders in Lab 4
  - Multiplication in lab 5
  - If lab has a state machine, know generally what the states are and do

Experiment 1: Static Hazards



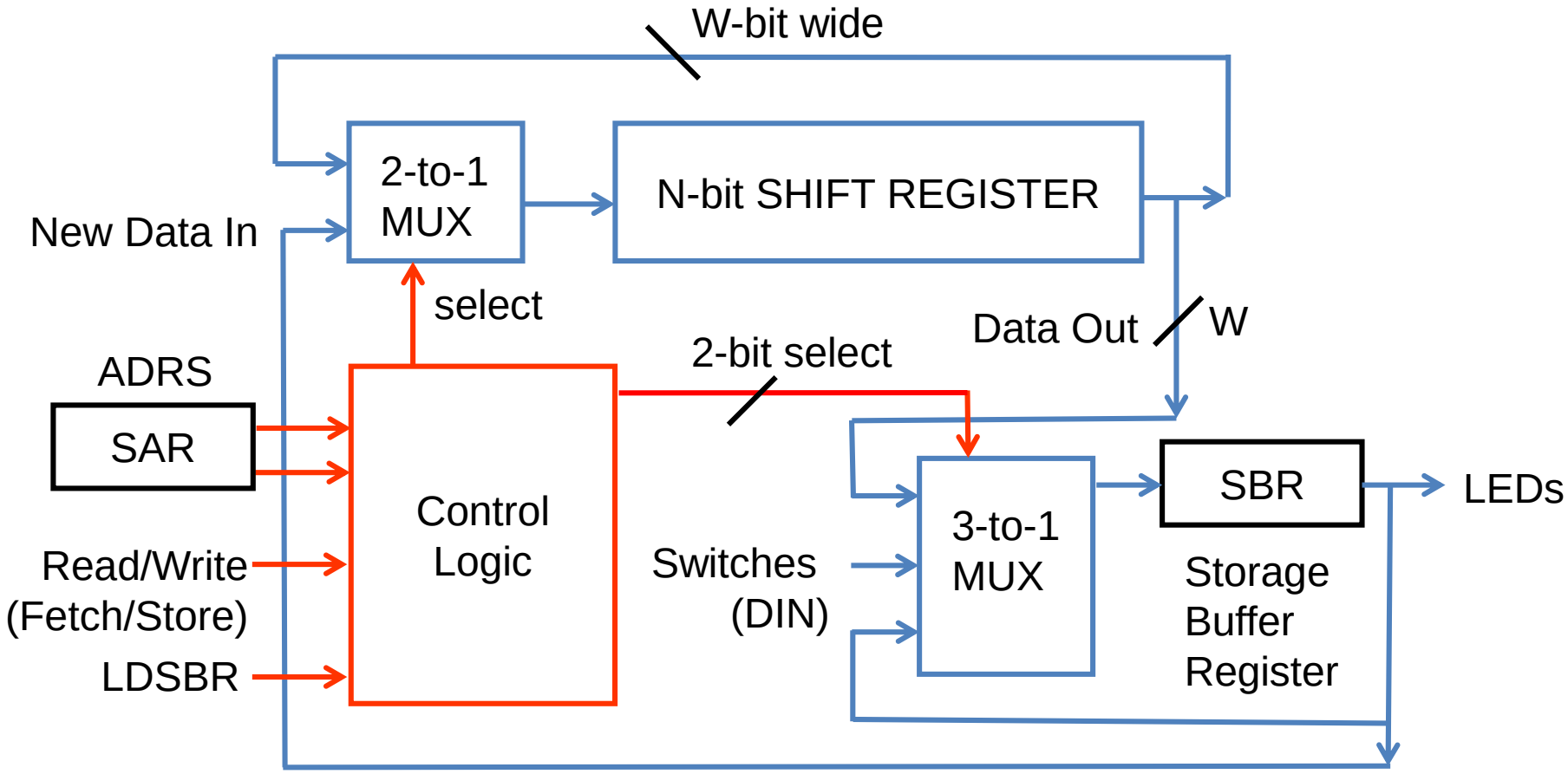


To avoid Static-1 hazard in an AND-OR (SOP) circuit, cover all adjacent min-terms in the K-map.
In this example, add the term AC

- What is the final Boolean function for glitch-free circuit?
  - Z = B'C + BA + AC

Experiment 2: Data Storage with TTL

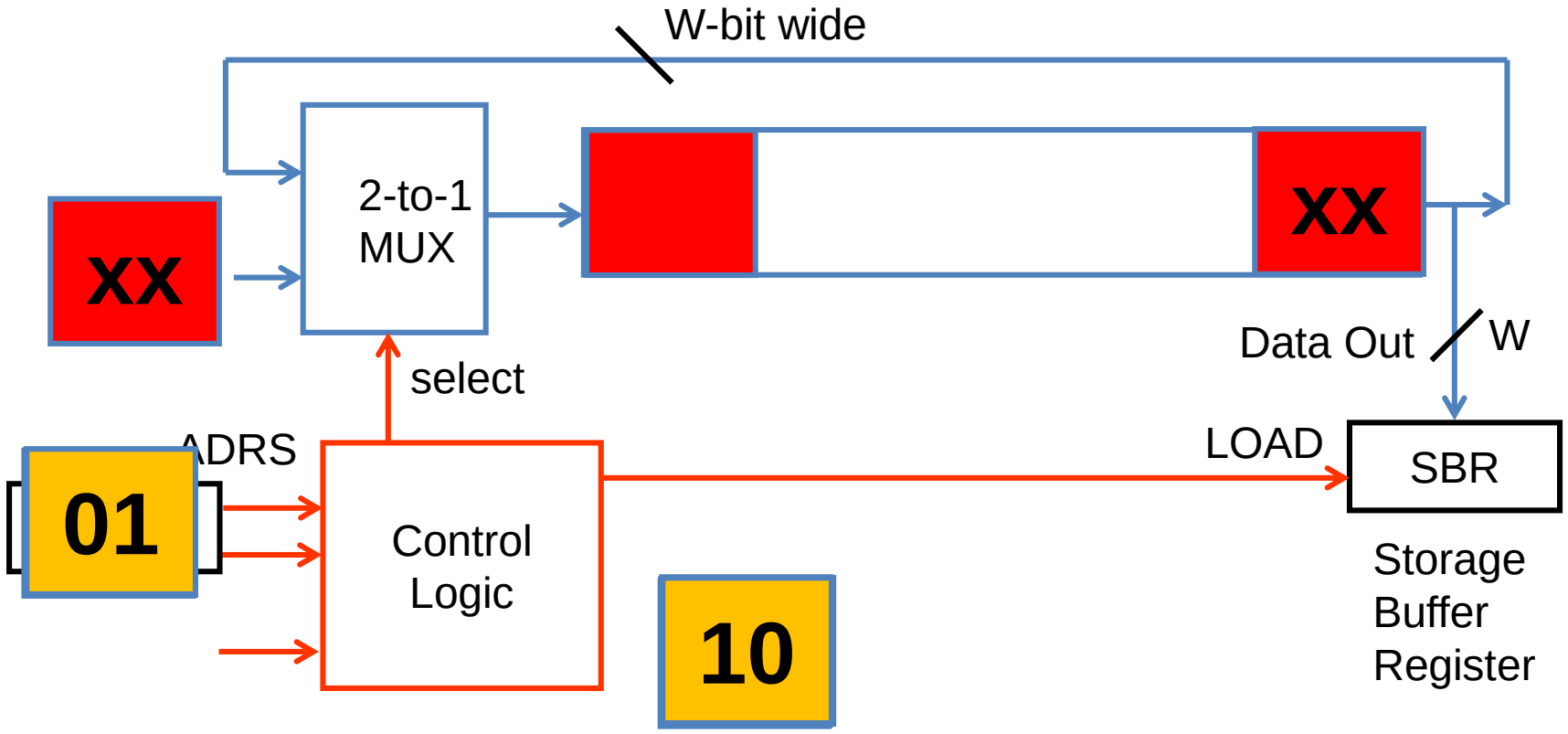
Use two 74LS194 shift-registers
Serial Input and Output
Circulating data

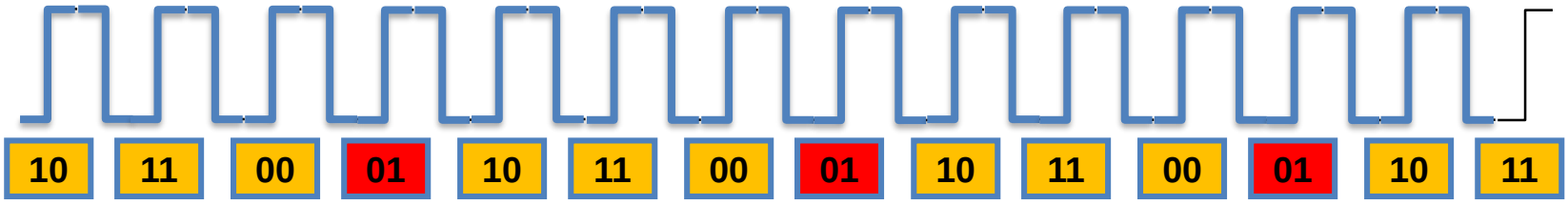


Experiment 2: Inputs and Outputs

- FETCH (Switch)
  - When FETCH is high, the value in the data word specified by the SAR is read into the SBR.
- STORE (Switch)
  - When STORE is high, the value in the SBR is stored into the word specified by the SAR.
- SBR1, SBR0 (Flip-flops)
  - The data word in the SBR; either the most recently fetched data word or a data word loaded from switches
- SAR1, SAR0 (Switches)
  - The address, in the SAR, of a word in the storage
- DIN1, DIN0 (Switches)
  - Data word to be loaded into SBR for storing into storage
- LDSBR (Switch)
  - When LDSBR is high, the SBR is loaded with the data word DIN1, DIN0

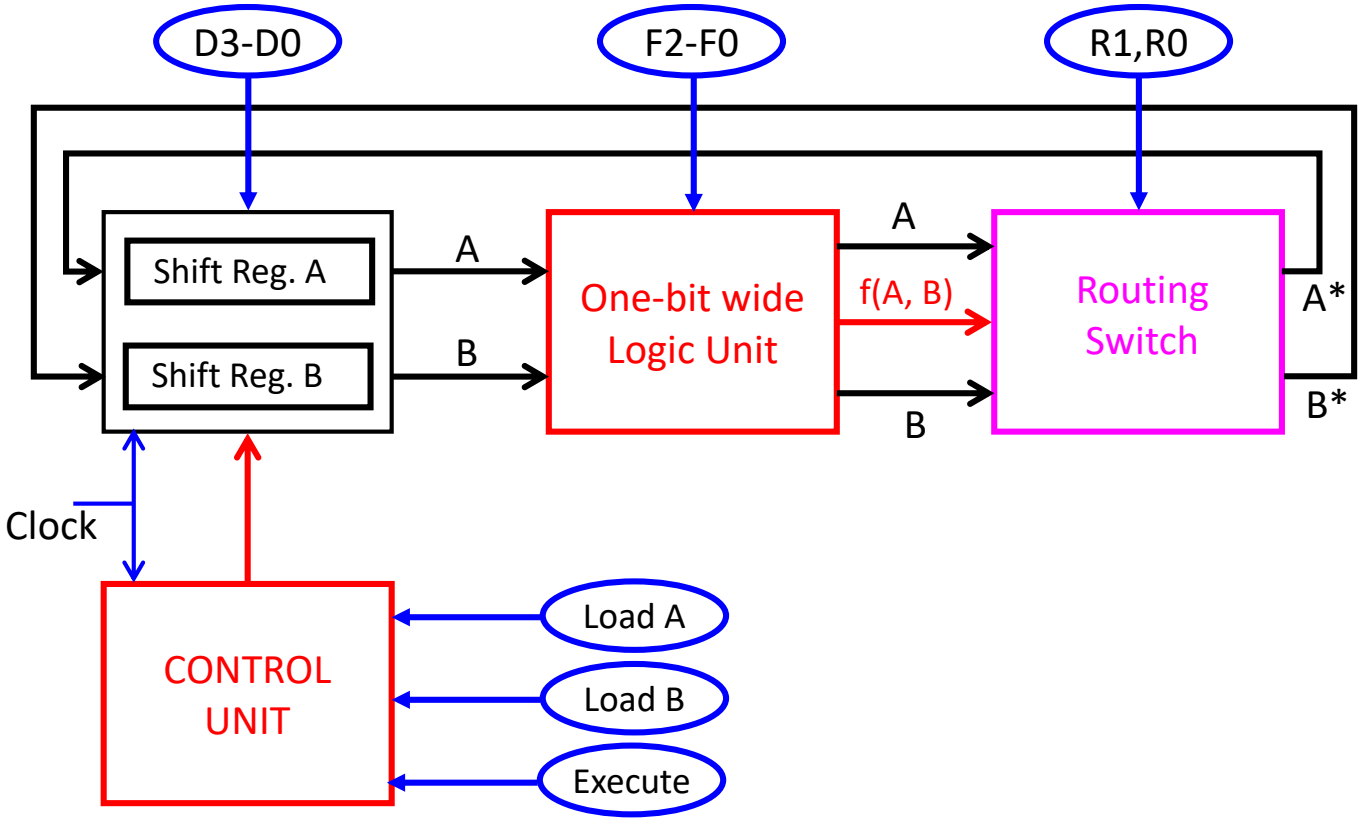
Experiment 2: Operation of Circuit





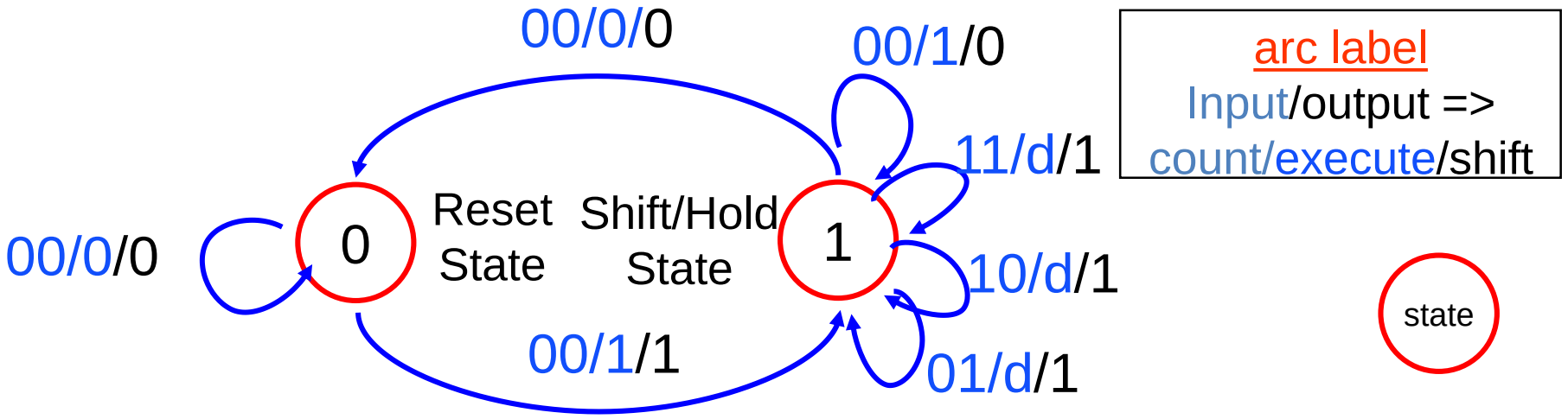
Experiment 3: 4-bit Serial Processor

- A Bit-Serial Logic Processor
  - Two 4-bit registers A, B
  - A or B also serve as a destination register.
  - E.g., AND A, B, A /* A AND B=>A */



Experiment 3: State Machine (Mealy)

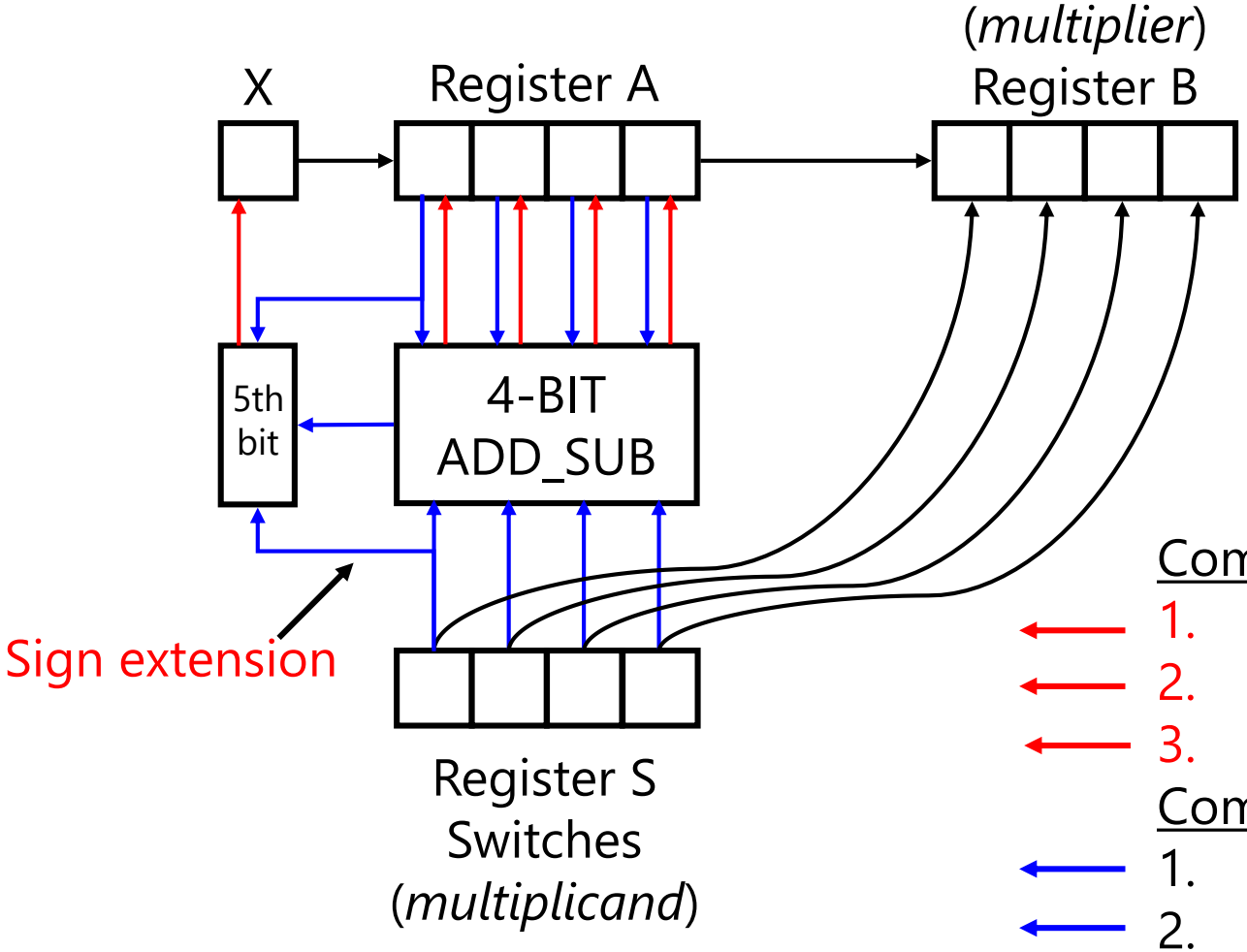
- Arcs are labeled with both inputs & outputs
- Fewer states than Moore machine (outputs depend on both state and inputs)



Experiment 4: Adders in SystemVerilog

- Experiment 4 had 3 different types of adders
  - Carry ripple adder
  - Carry look-ahead adder
  - Carry select adder
- Know how they work and what are tradeoffs
  - Carry ripple adder – smallest and simplest
  - Carry look-ahead adder – generate P and G bits beforehand to predict carry
  - Carry select adder – pre-compute both carry cases and select correct one
- Know block diagrams for all 3 adders
- Understand how CLA and CSA speed up from CRA!

Experiment 5: Multiplier in SV



Commands (push buttons)

1. ClearXA_loadB
2. RUN (Execute)
3. Reset

Commands (control)

1. SHIFT XAB
2. ADD (A+S → XA)
3. SUB (A–S → XA)

SHIFT, ADD *and* SUB *are derived signals from State Controller. They last only one clock cycle*

State Machine: States (4 bit)

state: Action <start when RUN switch is ON>

S0: XA ← A + M•S <if M=1 then Add S to the partial product>

S1: SHIFT XAB <arithmetic right shift >

S2: XA ← A + M•S [fn<='0'; if M=1 then loadX <='1'; loadA <='1';]

S3: SHIFT XAB

S4: XA ← A + M•S

S5: SHIFT XAB

S6: XA ← A - M•S <if M=1 then fn<='1"; Subtract S from partial product>

S7: SHIFT XAB <final product in Register AB>

S8: Wait Until RUN switch returns to 0

Midterm Review – Other Topics

- TTL Debugging - General Guide, Driving LEDs, Switches, Contact Bounce, Debouncing
- FPGA Fundamentals – Synthesis vs Simulation, FPGA organization (very broadly), Basic SystemVerilog coding, Procedure blocks (always_ff, always_comb), Timing
- Common HDL patterns and errors
  - Assignment types (blocking vs non-blocking – how they are treated differently in simulation)
  - Different array types (packed, unpacked)
  - Advanced timing, glitches, synchronizers, metastability
  - State machines, one-always & two-always coding styles

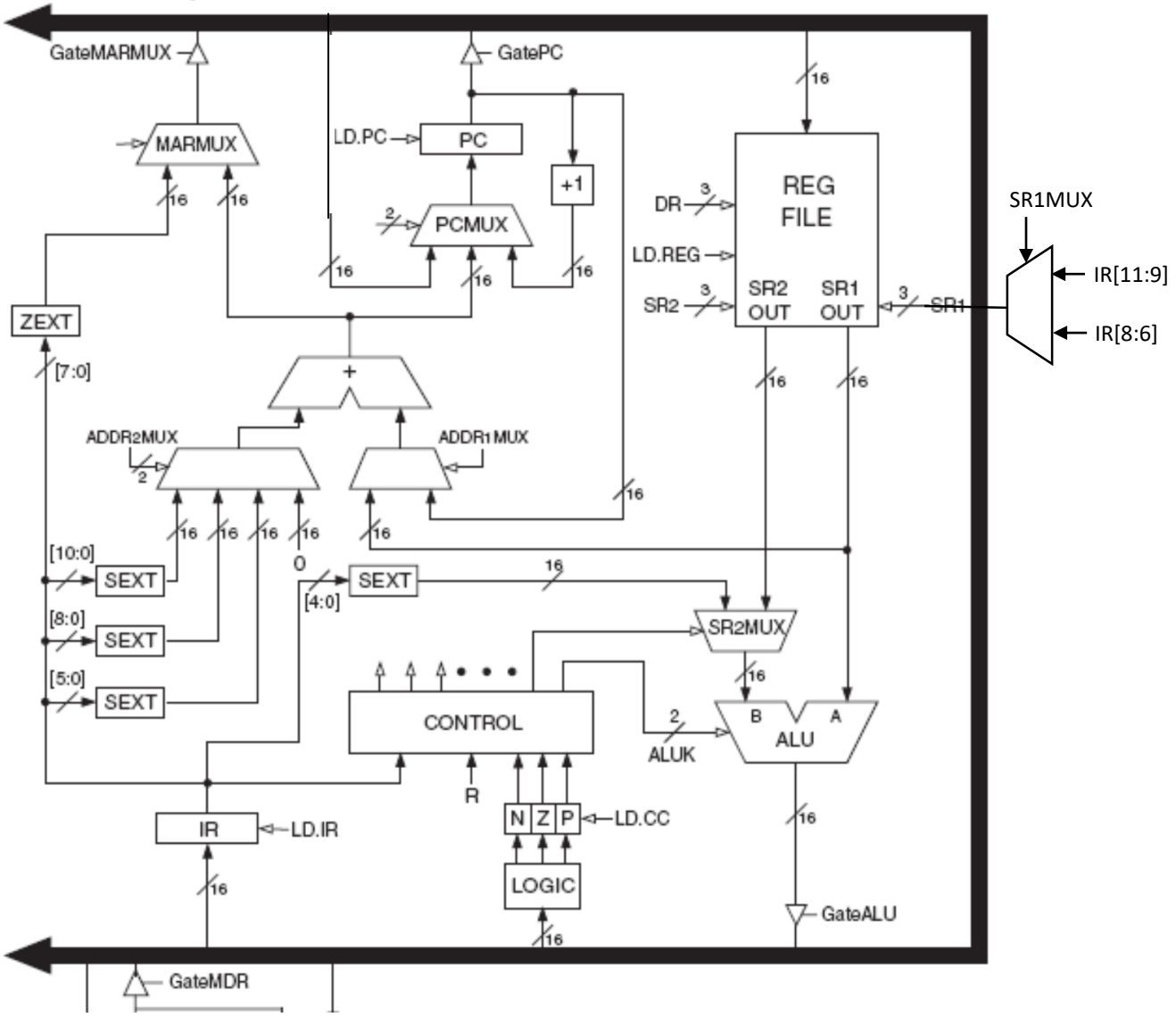
Week 2 Demo

- Basic I/O Test 1. (1.0 point)
  - Displays LED on switches (continuous)
- Basic I/O Test 2. (1.0 point)
  - Displays LED on switches, pause and wait for continue
- Self-Modifying Code Test. (1.0 point)
- XOR Test (1.0 point)
- Multiplication Test. (1.0 point)
- Sort Test. (1.0 point)
- Correct "Act Once" Behavior. (1.0 point)
  - Note: DE2 Control Panel application is unreliable - program at least ~~two~~ many times
  - Alternative: Use the SD Card SRAM programmer (reliable, only need to program once)

SLC-3 ISA – Subset of LC-3 ISA

Instruction	Instruction(15 downto 0)																Operation
ADD	0001			DR			SR1	0	00			SR2					R(DR) ← R(SR1) + R(SR2)
ADDi	0001			DR			SR	1	imm5								R(DR) ← R(SR) + SEXT(imm5)
AND	0101			DR			SR1	0	00			SR2					R(DR) ← R(SR1) AND R(SR2)
ANDi	0101			DR			SR	1	imm5								R(DR) ← R(SR) AND SEXT(imm5)
NOT	1001			DR			SR	111111									R(DR) ← NOT R(SR)
BR	0000			N	Z	P	PCoffset9										if ((nzp AND NZP) != 0) PC ← PC + 1 + SEXT(PCoffset9)
JMP	1100			000			BaseR	000000									PC ← R(BaseR)
JSR	0100			1	PCoffset11												R(7) ← PC + 1; PC ← PC + 1 + SEXT(PCoffset11)
LDR	0110			DR			BaseR	offset6									R(DR) ← M[R(BaseR) + SEXT(offset6)]
STR	0111			SR			BaseR	offset6									M[R(BaseR) + SEXT(offset6)] ← R(SR)
PAUSE	1101	ledVect12															LEDs ← ledVect12; Wait on Continue

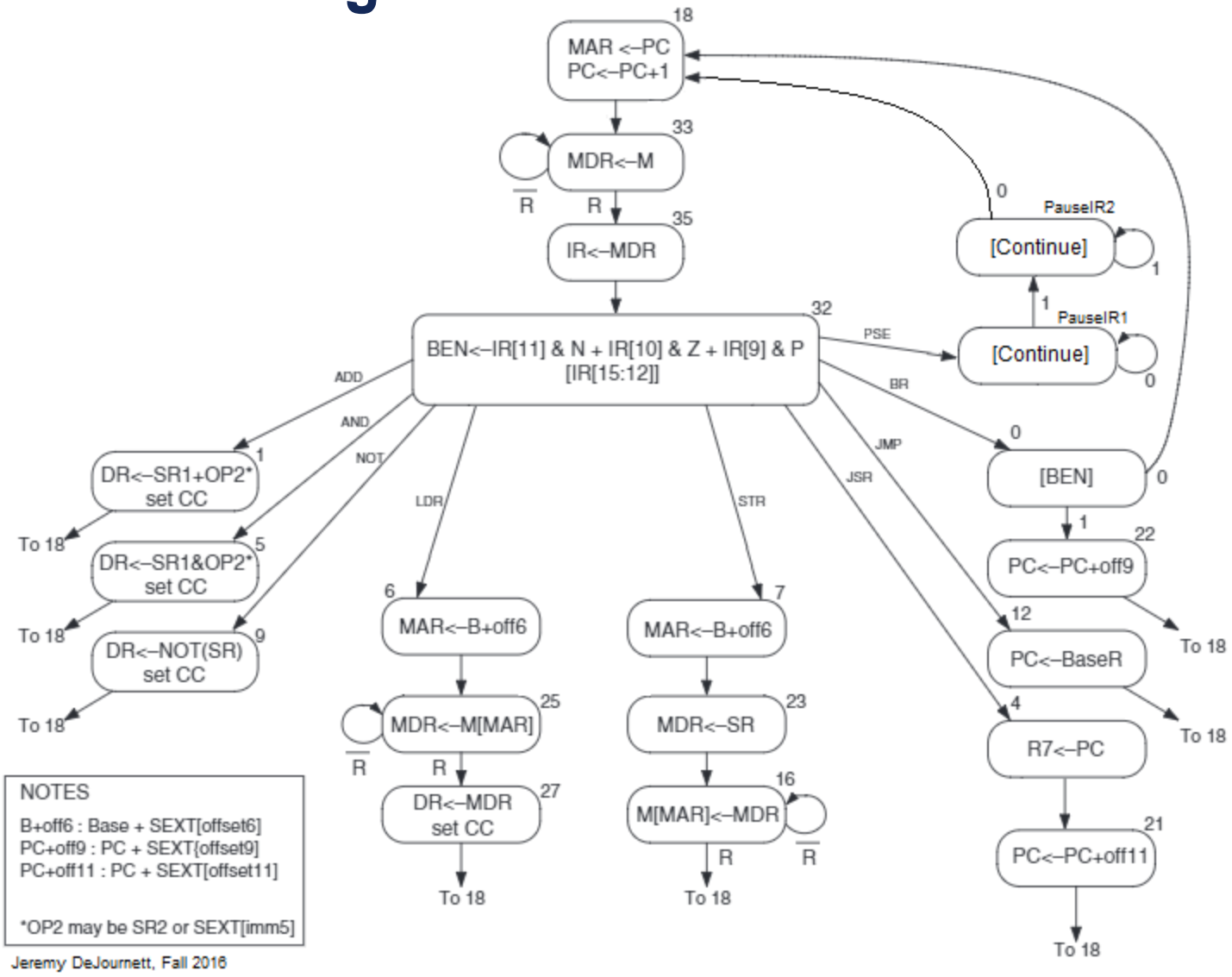
SLC-3 Block Diagram



MDR is down here

For complete diagram check out the online mater

Updated State Diagram



Jeremy DeJournett, Fall 2016

FETCH Phase

- state1: MAR ← PC
- state2: MDR ← M(MAR); -- *assert Read Command on the RAM*
- state3: IR ← MDR;
  PC ← PC+1; -- "+1" inserts an incrementer/counter instead of an adder.
  Go to decode

<u>More details:</u>

- MAR ← PC; MAR = memory address to read the instruction from
- MDR ← M(MAR); MDR = Instruction read from memory
- IR ← MDR; IR = Instruction to decode
- PC ← (PC + 1)

EXECUTE Stage for Load/Store

LOAD:

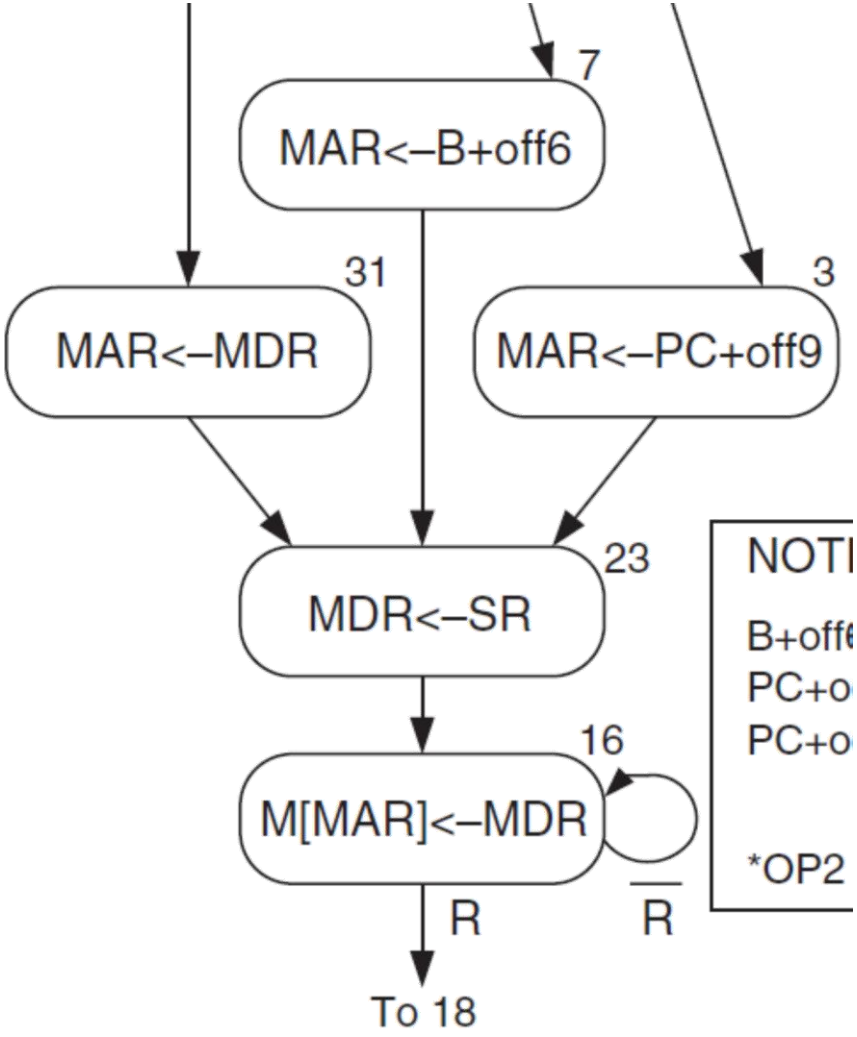
- Load_state1: MAR ← (BaseR + SEXT(offset6)) from ALU & AGU
- Load_state2: MDR ← M(MAR); -- *assert Read Command on the RAM*
- Load_state3: R(DR) ← MDR;

STORE:

- Store_state1: MAR ← (BaseR + SEXT(offset6)) from ALU & AGU
- Store_state2: MDR ← R(SR)
- Store_state3: M(MAR) ← MDR; -- *assert Write Command on the RAM*

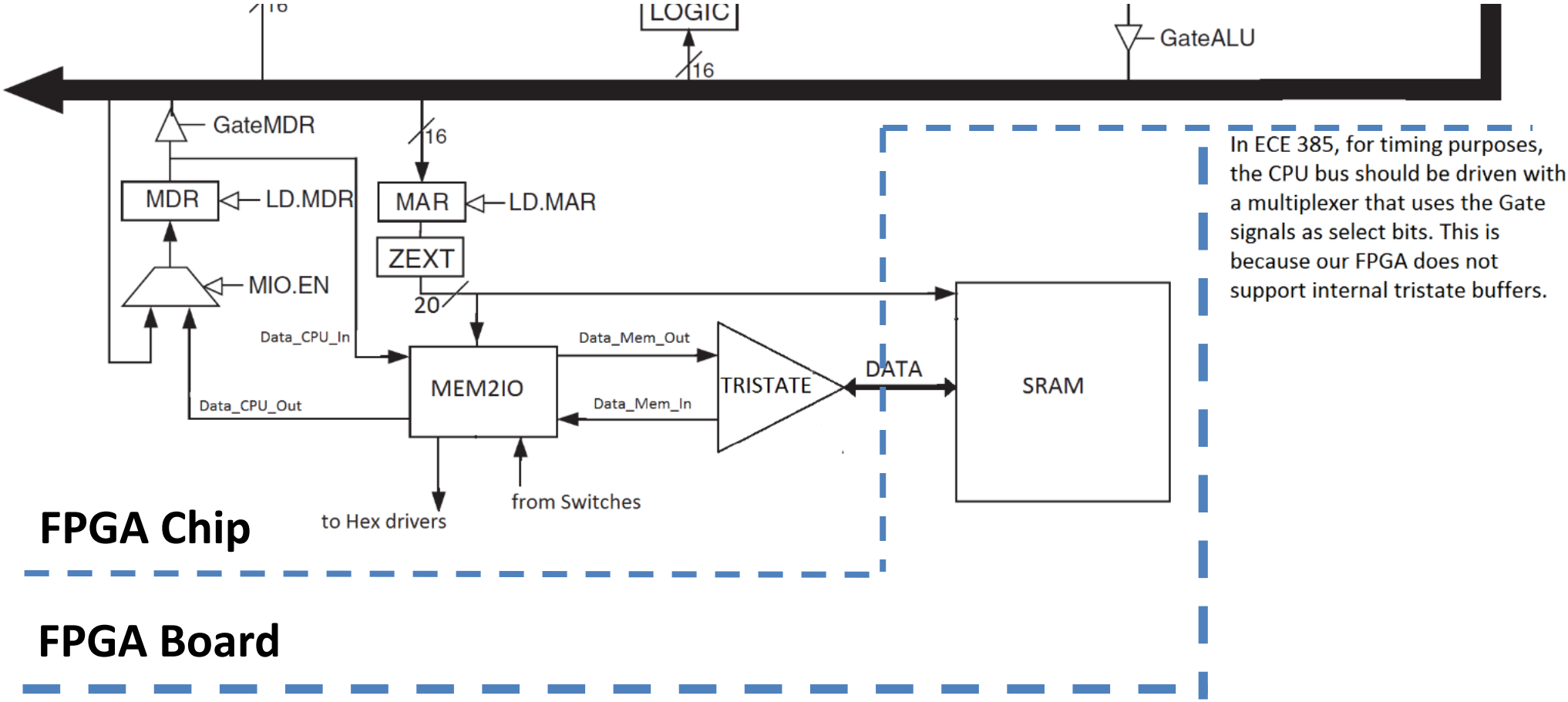
State Diagram: Store

- Store Decodes into state 7
- What is R?
- How many cycles are in R?



CPU to SRAM Configuration

- CPU to SRAM with Physical Memory



Interfacing: I/O addressing

- A microprocessor communicates with other devices using some of its pins
  - Port-based I/O (parallel I/O)
    - Processor has one or more N-bit ports
    - Processor's software reads and writes a port just like a register
    - E.g., P0 = 0xFF; v = P1.2; -- P0 and P1 are 8-bit ports
    - Often has special instructions for I/O
  - Bus-based I/O
    - Processor has address, data and control ports that form a single bus
    - Communication protocol is built into the processor
    - A single instruction carries out the read or write protocol on the bus
    - Often looks the same as Load/Store

Memory Mapped Bus-Based I/O

- We are implementing memory mapped bus based I/O
- Primary function of the Mem2IO block
  - Detects load from address 0xFFFF as a load from switches (instead of real SRAM word 0xFFFF)
  - Detects store to address 0xFFFF as store out to HEX displays (stores this value in internal register and displays it instead of storing to SRAM)
- Note:
  - Don't need to implement any special instructions for I/O (Load/Store sufficient)
  - Things get potentially messy if CPU has cache (won't have to deal with for now)

A Reminder About Endianness

- All LC-3 CPUs (LC-3, LC-3b, SLC-3) are little endian
- Little Endian -> least significant byte stored at lowest address
- Big Endian -> most significant byte stored at lowest address
- This may result in bytes appearing to be in wrong order when viewing on hex editor

Implementing Register File

- Can use 8 16-bit registers (extend reg8 to reg16)
- What goes to D (data input port), what goes to DR?
- What about Q1, Q2 (data output ports), SR1, SR2?
- Alternatively, can use unpacked array of packed logic:
  - `logic [15:0] reg_file [8];`
- Different approaches illustrate difference between behavioral HDL and structural HDL

Understanding Test Programs

- Make sure you understand the test programs for Week 2
- Note: you can write your own test programs by using test_memory.sv and slc3_2.sv
  - slc3_2.SV is a SystemVerilog library which has un-synthesizable functions which act as an assembler
  - This allows test_memory.sv to have assembly language syntax
  - Note: all test_memory.sv has same contents as RAM image

```
mem_array[   0 ] <=    opCLR(R0)                ;       // Clear the
   register so it can be used as a base
mem_array[   1 ] <=    opLDR(R1, R0, inSW)      ;       // Load switches
mem_array[   2 ] <=    opJMP(R1)                ;       // Jump to the
   start of a program
```

Basic I/O Test 1

- Starts at address 0x03
  - Should be able to set switch to 3 and reset to run Basic I/O Test 1
  - Note – all SLC-3 addresses are word addresses
  - LC-3 addresses are byte addressable (hence PC+2 on some other LC-3 materials)
- I/O Test displays contents of switches on HEX displays
  - Tests load from 0xFFFF (-1 sign extended – maps to switches)
  - Tests store to 0xFFFF (-1 sign extended – maps to HEX displays)

```
mem_array[   3 ] <=    opLDR(R1, R0, inSW)       ;       // Load switches
mem_array[   4 ] <=    opSTR(R1, R0, outHEX)     ;       // Output
mem_array[   5 ] <=    opBR(nzp, -3)             ;       // Repeat
```

Another Test Program (Basic I/O Test 2)

- Make sure you understand the test programs for Week 2 Basic I/O Test 2
- The first pause instruction (checkpoint 1) will ask for input on the switches
- The second and subsequent pauses will display x02 on the LED, and ask for input and report that an output is present on the HEX display
- When operating correctly, each press of the Continue button will transfer the value from the switches to the HEX display, but the HEX display will not change until Continue is pressed

```
mem_array[   6 ] <=    opPSE(12'h801)           ;       // Checkpoint 1 -
   prepare to input
mem_array[   7 ] <=    opLDR(R1, R0, inSW)      ;       // Load switches
mem_array[   8 ] <=    opSTR(R1, R0, outHEX)    ;       // Output
mem_array[   9 ] <=    opPSE(12'hC02)           ;       // Checkpoint 2 -
   read output, prepare to input
mem_array[  10 ] <=    opBR(nzp, -4)            ;       // Repeat
```