Lecture 14: Microprogramming and Exceptions

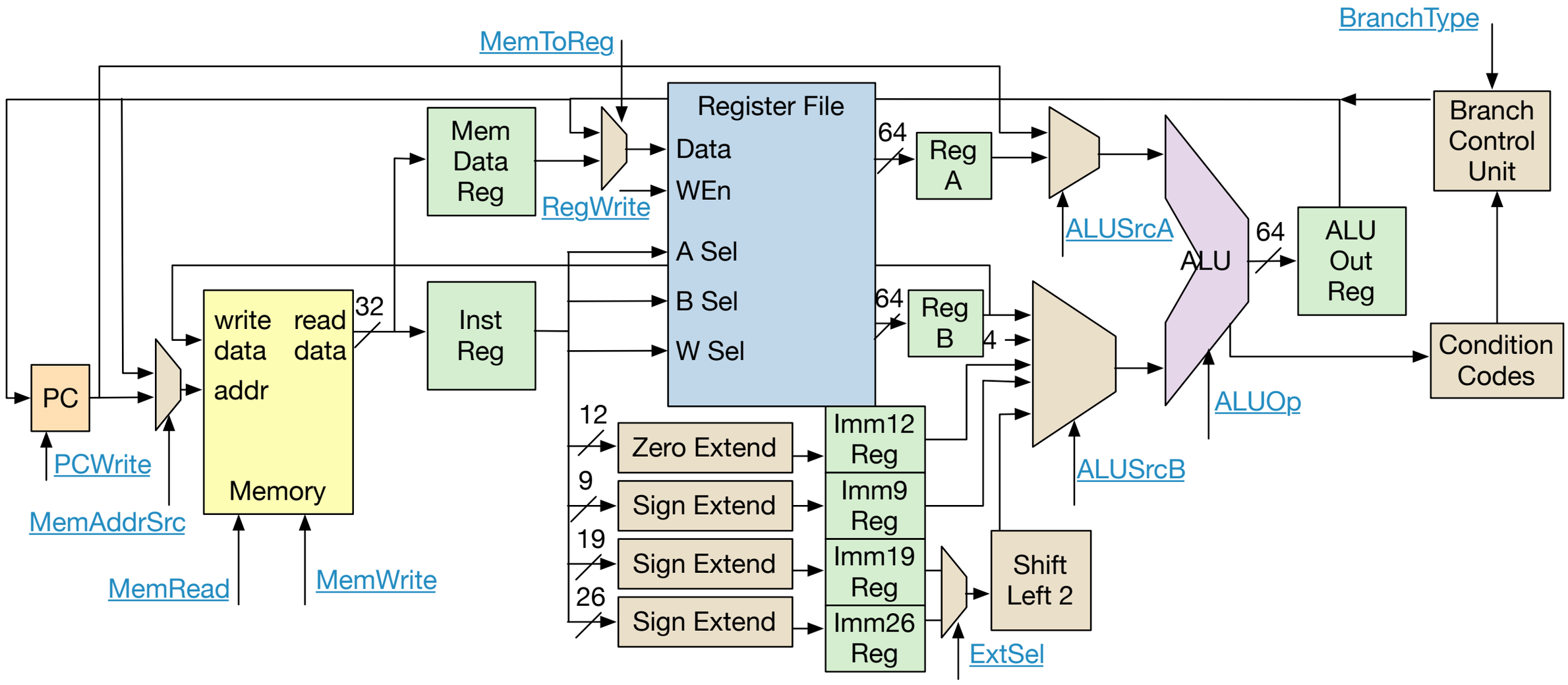
Spring 2024
Jason Tang

Topics

- Microprogramming control
- Processor exceptions
- Exception handling

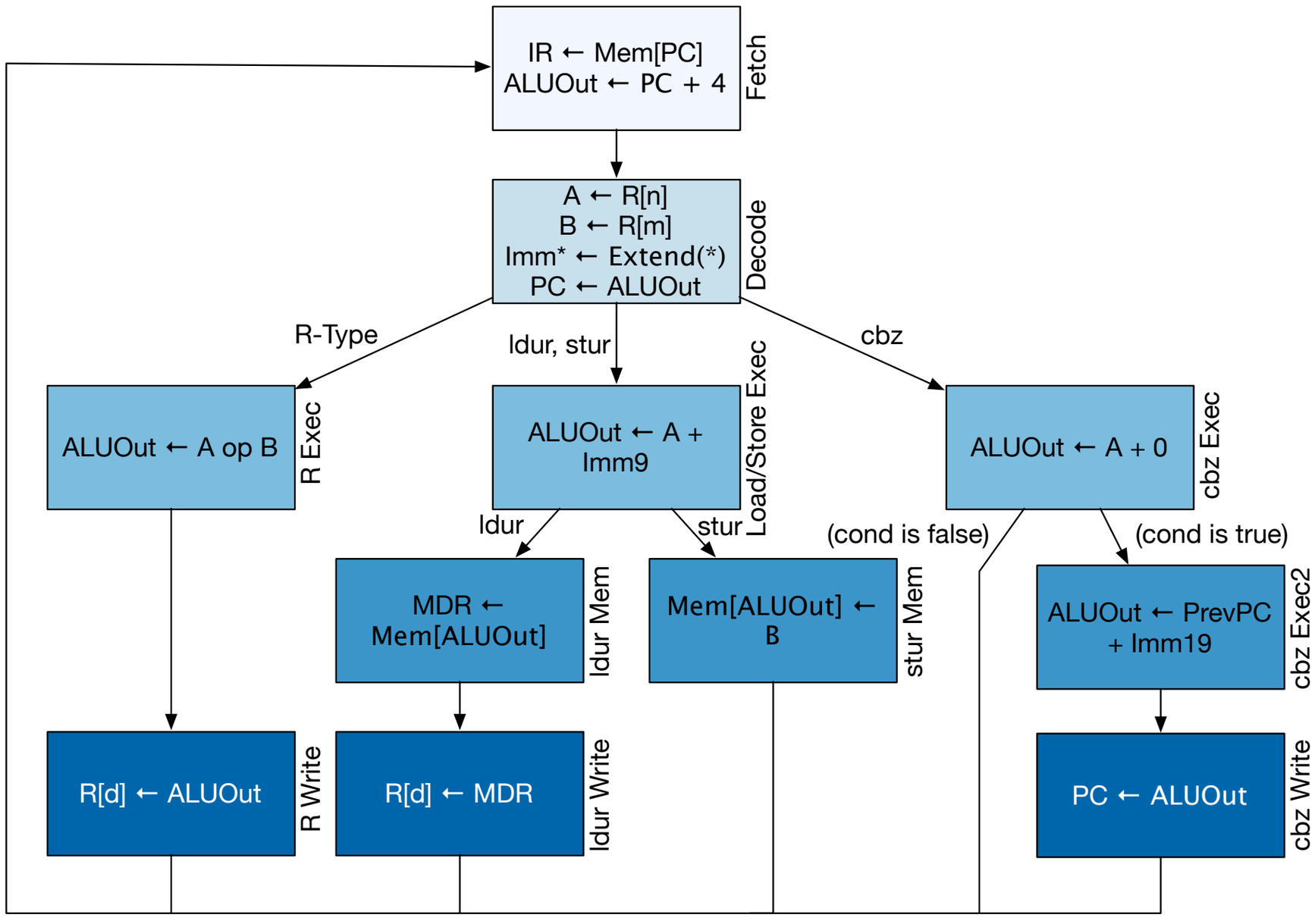
Mostly Complete Multi-Cycle Datapath

- Other than branching, this datapath handles basic ARMv8-A instructions



Multi-Cycle Finite State Machine

- Note this is a finite state machine (FSM)

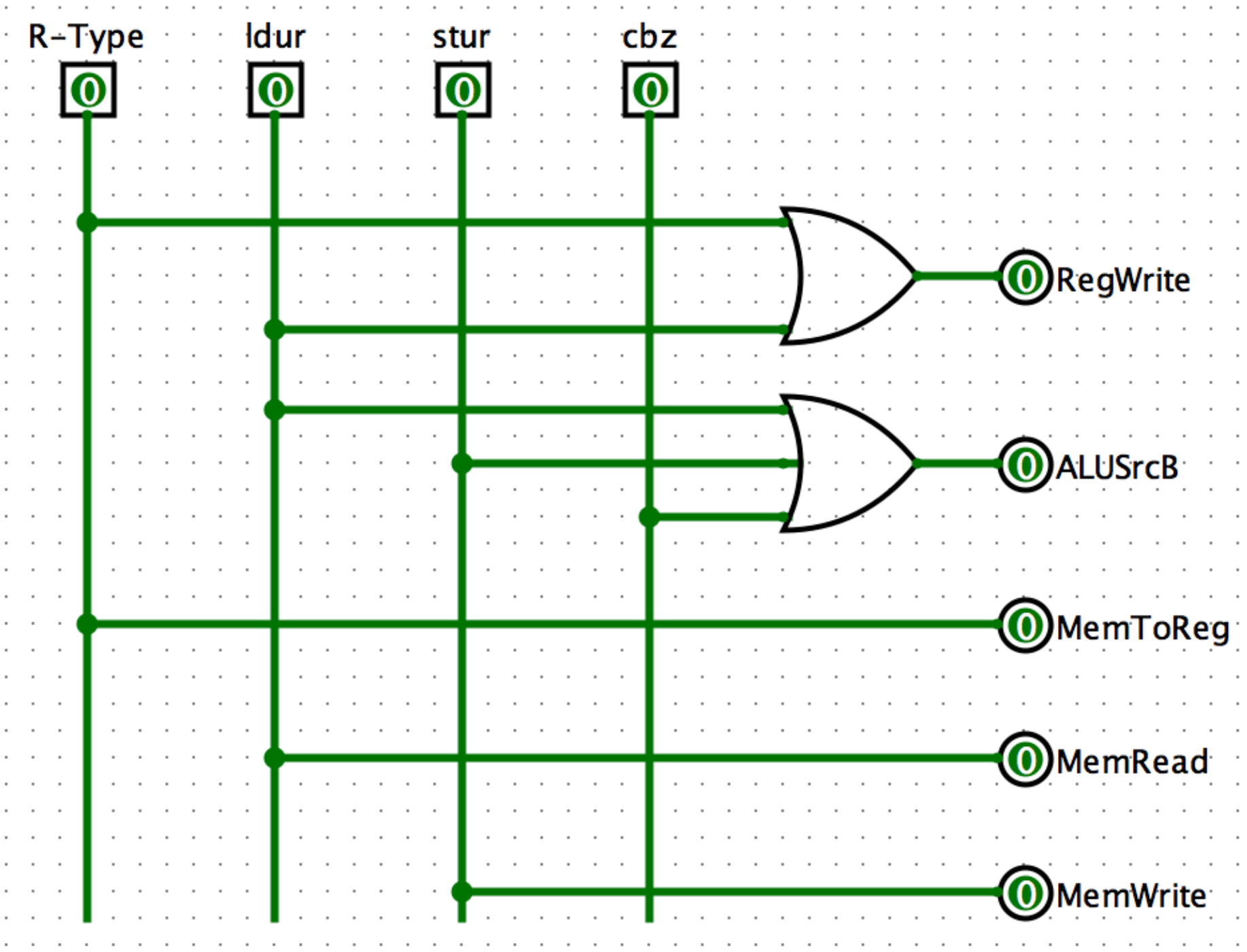


Microprogramming

- Instead of implementing FSM as a giant truth table, describe controls as a series of simpler microinstructions:
  - Defines set of datapath control signals to assert
  - Defines which microinstruction to execute next, based upon current instruction
- Microinstructions usually stored in a ROM or a PLA
  - ROMs are easier to change (and reprogrammable), but PLAs can be faster
  - EPROMS can be patched, to fix processor bugs

Reverse Engineering x86 Processor Microcode, https://www.syssec.rub.de/media/emma/veroeffentlichungen/2017/08/16/usenix17-microcode.pdf

Example ARMv8-A PLA Control



Microprogramming Syntax

Label	ALUSrcA	ALUSrcB	ALUOp	MemRead	MemWrite	PCWrite	Sequencing
Fetch	PC	4	add	Read PC			Seq

- Each line of microprogram describes a state, and which values to send to control lines
  - A blank column is a don't care, either a zero is written (for control lines) or any value (for a mux selector)
- Each line also includes a sequence field, to indicate which state to go next
  - In this syntax, the keyword `seq` means to proceed to following line

Conditional Sequencing

Label	ALUSrcA	ALUSrcB	ALUOp	MemRead	MemWrite	PCWrite	Sequencing
Decode						ALUOut	Dispatch 1

- When Sequencing field is not **`seq`**, then lookup next state based upon the instruction register's contents

- In this case, search through a subtable named *Dispatch 1* to determine next state

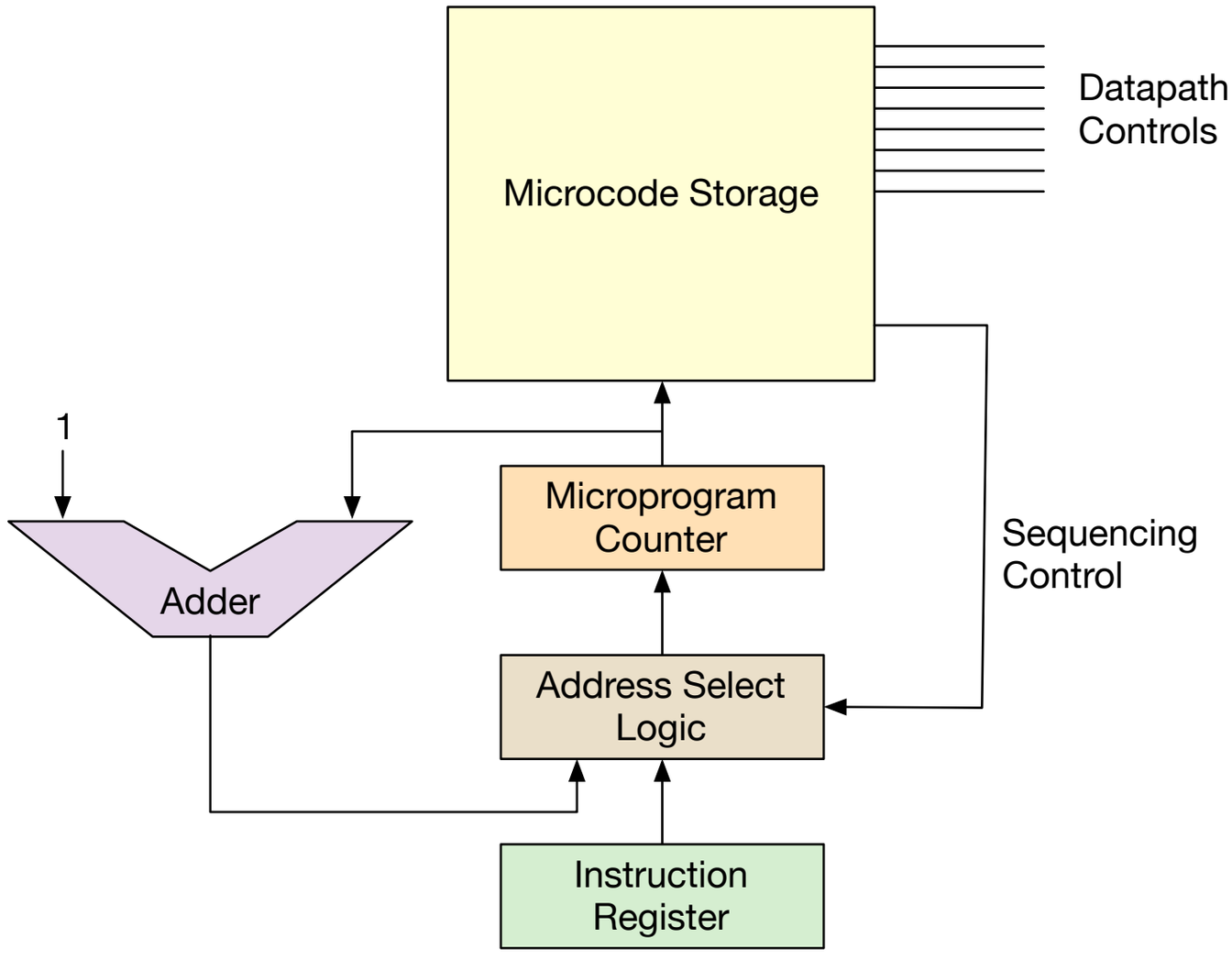
- Decode instruction based upon if it is an R-Type, **`ldur`**, **`stur`**, etc.

Partial ARMv8-A Microprogram

Label	ALUSrcA	ALUSrcB	ALUOp	MemAddrSrc	MemToReg	PCWrite	Sequencing
Fetch	PC	4	add	PC			Seq
						ALUOut	Dispatch 1
RType	RegA	RegB	func				Seq
					ALUOut		Fetch
Load/ Store	RegA	imm9	add				Dispatch 2
ldur				ALUOut			Seq
					MDR		Fetch
stur				ALUOut			Fetch

- Similar to normal programming, a microcode assembler ensures that for every state, conflicting signals are not asserted

Implementing Microcode



- Microcode controller looks and behaves similar to a full-scale processor

Methods for Control Implementation

	Hardwire Control	Microprogramming
Initial Representation	Finite state diagram	Microprogram
Sequencing Control	Explicit next-state function	Microprogram counter + dispatch ROMs
Logic Representation	Logic equations	Truth tables
Implementation Technique	Programmable logic array	Read-only memory

- For each row, either column A or B could be chosen
  - Traditional hardwired control prefer left column
  - Microprogrammed control prefer right column

Exceptions

- Multiple definitions for “interrupts” and “exceptions”
- As per textbook’s authors,
  - Exception: any unexpected change in control flow, regardless of internal or external cause
  - Interrupt: an exception that is caused by an external event

Types of Exceptions

Event	Source	ARMv8-A Terminology
System reset	External	Exception
I/O device request	External	Interrupt
Request operating system resource from user program	Internal	Exception
Floating-point arithmetic overflow/ underflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunction	Either	Exception or Interrupt

- Intel refers to all of the above as “interrupts”

- Regardless of name, hardware implementations are similar

Exceptions Overview

- CPU stores current PC when exception is detected
    - On ARMv8-A, hardware writes address to Exception Link Register (ELR)
- CPU transfers control to a interrupt service routine (ISR), through the interrupt vector table
    - Depending upon architecture, other registers may be preserved
- Software then resolves exception
- Software finally returns from ISR, causing CPU to restore registers and resume processing at the saved PC

Vectored Exceptions

- Each exception has a unique numeric code
    - Example: on x86-64, *undefined instruction* is exception number 06h
- For some architectures, that exception number is written to a special register when the hardware detects the exception
    - On ARMv8-A, hardware writes to Exception Syndrome Register (ESR)
- Then, depending upon the type of exception, the CPU jumps to an address relative to an interrupt base address

Exception Handling

- During exception handling, hardware preserves some registers; software is responsible for saving additional registers it needs

- ISR then handles exception
  - Example: if a program divides by zero, then OS terminates that program

- ISR restores any registers it modified

- ISR finally invokes a special instruction that returns from exception handling, causing hardware to restore PC and resume processing
  - On ARMv8-A, the **`eret`** instruction jumps to the address stored in ELR

ARM Architecture Reference Manual for A-profile architecture, section C6.2.121

Addressing Exception Handlers

- Traditional Vectored Interrupt (x86-64):
  - PC ← MEM[**`IV_Base`** + (**N** × *Vector_Size*)]
- Interrupt Vector Registers (PowerPC):
  - PC ← **`IVOR`*N***
- RISC Style (ARMv7, ARMv8-A):
  - PC ← **`IV_Base`** + (**N** × *Vector_Size*)
  - For ARMv7, *Vector_Size* is 4 bytes, to [usually] hold a branch instruction; for ARMv8-A, *Vector_Size* is 64 bytes and can hold entire ISR

ARM Cortex-A Series Programmer's Guide for ARMv8-A, section 10.4

ARMv7-A Exception Vector Table

Exception	Offset
Reset	0000_0000h
Undefined Instruction	0000_0004h
Supervisor Call	0000_0008h
Prefetch Abort	0000_000ch
Data Abort	0000_0010h
Hypervisor Trap	0000_0014h
IRQ interrupt	0000_0018h
Fast IRQ (FIQ) interrupt	0000_001ch

ARM Architecture Reference Manual vJ.a, section G.1.12.1

x86-64 Interrupt Table

- Interrupt Descriptor Table (IDT) is a special register that holds the starting address, within memory, to interrupt vector table

- IDT points to a table of 256 IDT descriptors:

```
struct IDTDescr {
   uint16_t offset_1; // offset bits 0..15
   uint16_t selector; // a code segment selector in GDT or LDT
   uint8_t ist;       // bits 0..2 holds Interrupt Stack Table
                      // offset, rest of bits zero.
   uint8_t type_attr; // type and attributes
   uint16_t offset_2; // offset bits 16..31
   uint32_t offset_3; // offset bits 32..63
   uint32_t zero;     // reserved
};
```

- When interrupt **N** occurs, the processor goes to IDT entry **N**, constructs a 64-bit address, then jumps to that address

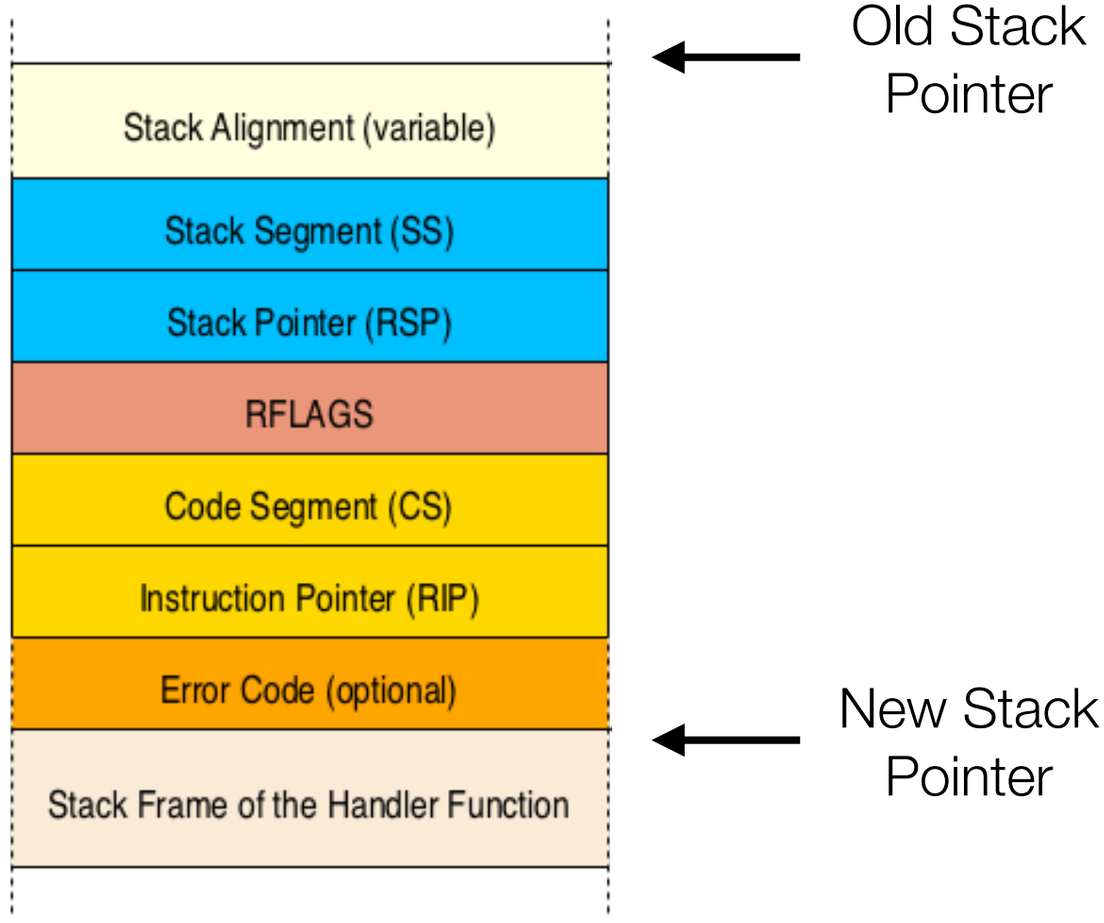
https://wiki.osdev.org/Interrupt_Descriptor_Table

x86-64 Exception Handling

- Hardware saves registers to memory, below the current value of the stack pointer

- CPU then jumps to address constructed by IDT

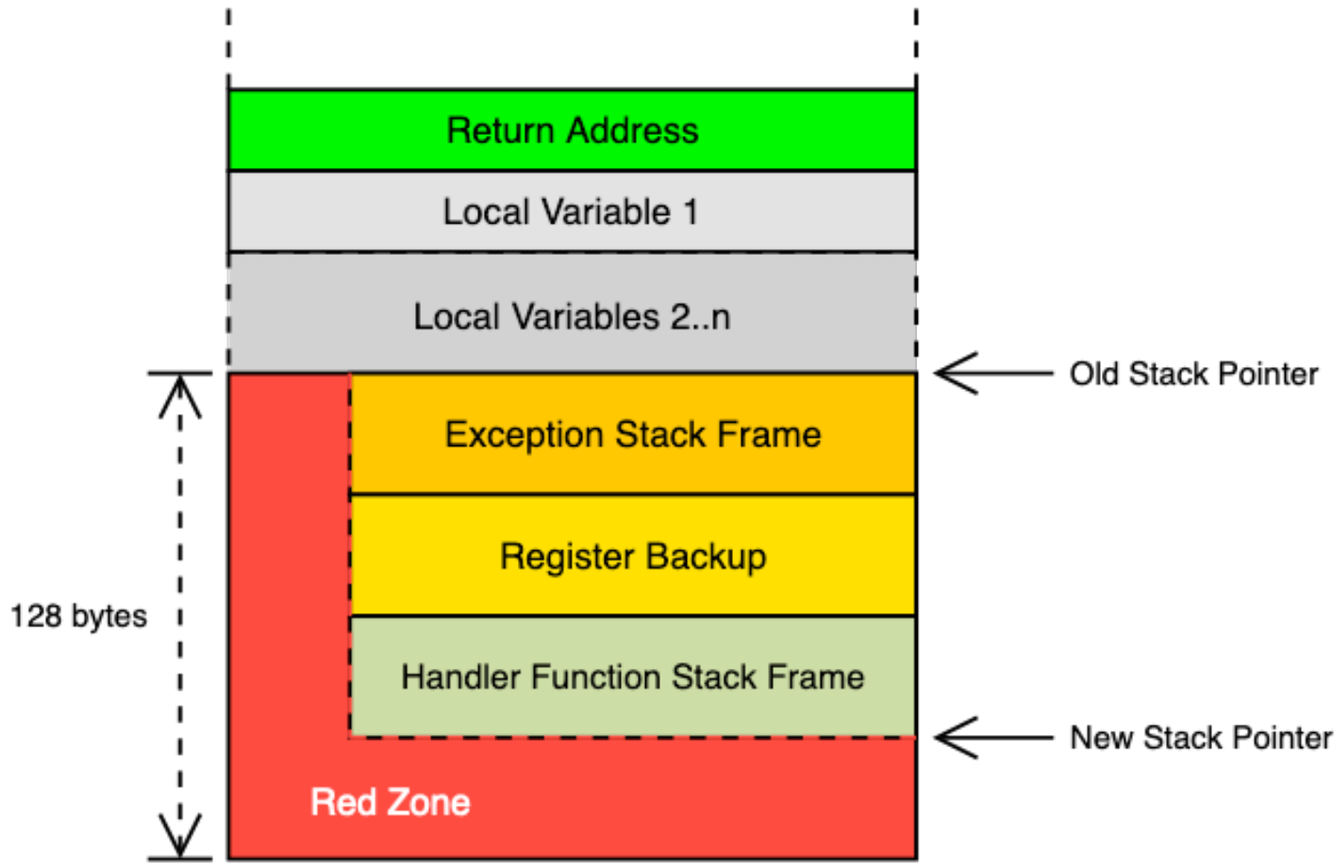
- When exception handler completes, it uses `iretq` to restore registers from the stack and resume processing



https://os.phil-opp.com/returning-from-exceptions/

x86-64 Red Zone

- As an optimization, a function may try to use space below the stack pointer as scratch space, without modifying the stack pointer



- But because x86-64 ISR's stack also uses the same memory, it has to avoid clobbering that data

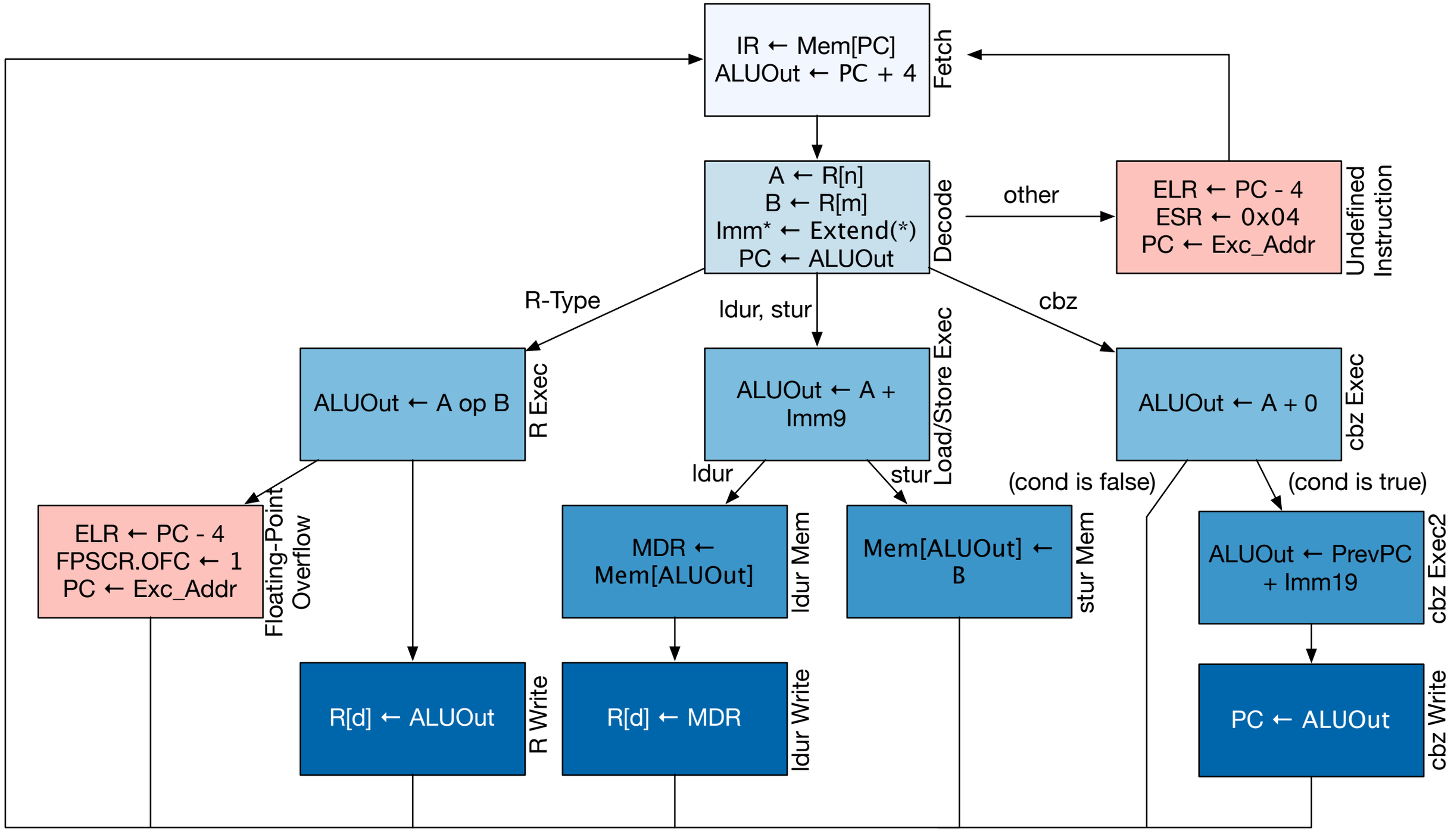
- Red Zone: Area below stack pointer that can safely be used by leaf functions, and will be untouched by ISR

https://os.phil-opp.com/returning-from-exceptions/

Detecting Exception Examples

- **Undefined Instruction**: detected when no next state is defined as a result of decoding instruction register
    - Shown symbolically as *“other”* in FSM / microprogram when opcode field does not match
- **Arithmetic Overflow**: detected when overflow bit is set in ALU’s condition code
- Because PC was probably incremented by the time an exception is detected, need to decrement saved PC (subtract by 4) before writing it to ELR

Exception Handling in Multi-Cycle Datapath



Special Purpose Registers

- ESR and ELR are examples of system registers (a type of special purpose register)

- Register file holds general purpose registers, not system registers
  - Usually, software cannot use system registers as ALU operands

- Special instructions used to interact with system registers
  - On ARMv8-A, use `mrs` to copy a system register's value to a GPR, `msr` to copy a GPR value back to a system register
  - Typically, hardware changes behavior immediately as a side effect of writing to the system register

Examples of ARMv8-A Special Purpose Registers

Register Name	Usage
CurrentEL	Holds current exception level
ELR_EL1	Holds return address when exiting EL1
FPCR	Floating-point control register
NZCV	ALU condition codes register
SP_EL1	Stack pointer for when entering EL1
SPSR_EL1	Saved program status register when entering EL1

ARM Architecture Reference Manual for A-profile architecture, section C5.2