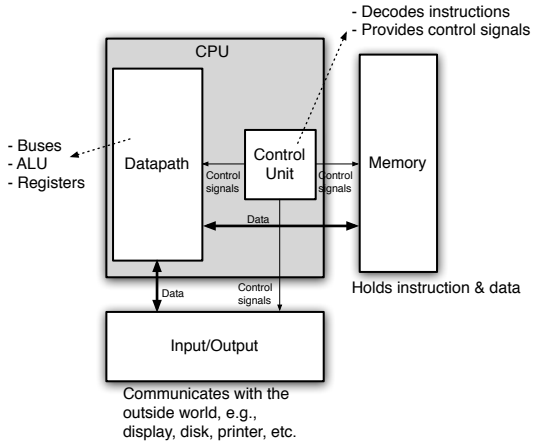
Chapter 3: Computer Organization Fundamentals

Prof. Ben Lee
Oregon State University
School of Electrical Engineering and Computer Science

Chapter Goals

- Understand the organization of a computer system and its components.
- Understand how assembly instructions are executed on the processor.

Computer Organization



Memory

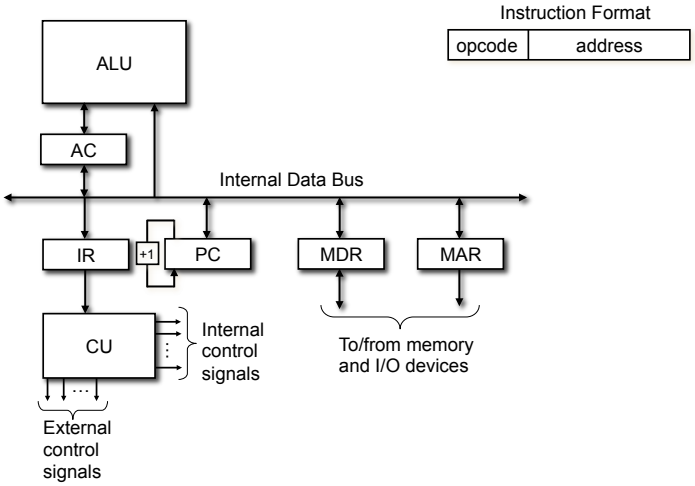
- Random Access Memory
- Holds instrutions (program) and data
  - Unified
  - Separate instruction and data memory
- Organized into consecutive addressable memory words.
- 1 memory word
  - Memory data size
  - Size of the information accessed by the CPU (CPU register size)
  - Manufacturer's definition

Registers

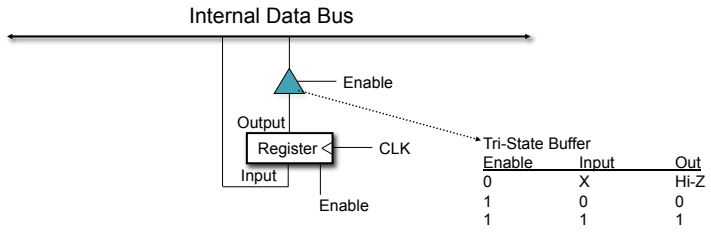
Some important registers:

- PC (Program Counter) – holds the address of the next inst. to be fetched from memory
- MAR (Memory Address Register) – holds the address of the next instruction or data to be fetched from memory.
- MDR (Memory Data Register) – hold the information (word) to be sent to/from memory.
- AC (accumulator) – a special register which holds the data to be manipulated by the ALU.
- IR (Instruction Register) – holds the instruction to be decoded by the Control Unit (CU).

A Pseudo-CPU



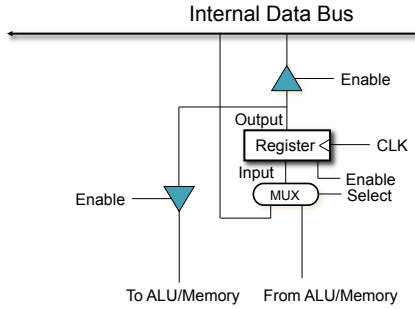
Bus-Register Connections



Tri-State Buffer

Enable	Input	Out
0	X	Hi-Z
1	0	0
1	1	1

Bus-Register Connections



Tri-State Buffer

Enable	Input	Out
0	X	Hi-Z
1	0	0
1	1	1

Fetch and Execute Cycle

- A series of steps (i.e., micro-operations) a computer takes to fetch *and* execute one instruction.
  - Each micro-operation requires a clock cycle.
- **Fetch and execute cycle => Instruction Cycle.**
- Number of micro-operations required to fetch an instruction is usually the same.
- Number of micro-operations required to execute each instruction differs depending on
  - Complexity of the instruction
    - e.g., Multiply takes longer than Add
  - Available hardware
    - e.g., Multiplier vs. no multiplier hardware

Fetch Cycle

- Need to describe what has to happen in each cycle.
- Will use register transfer operations to describe the movement of data.

Fetch Cycle

Step 1: MAR ← PC

Step 2: MDR ← M(MAR) ; Transfer the content of memory
; pointed to by MAR

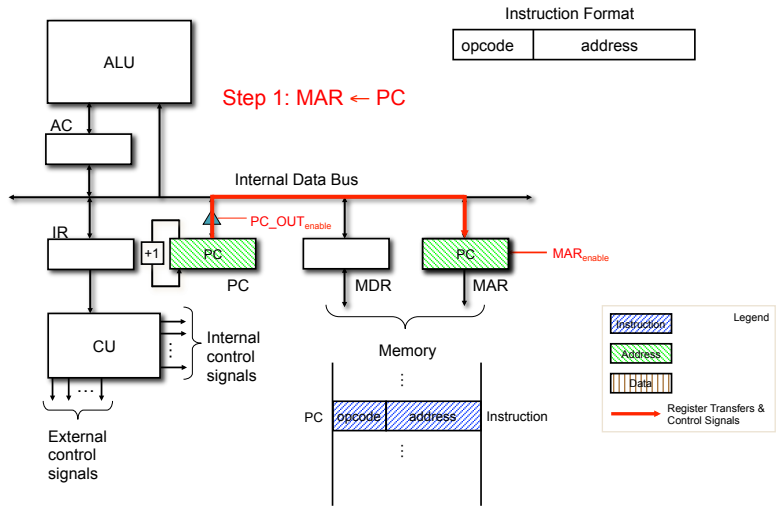
Step 3: IR ← MDR (opcode), MAR ← MDR (address)

Step 4: PC ← PC + 1

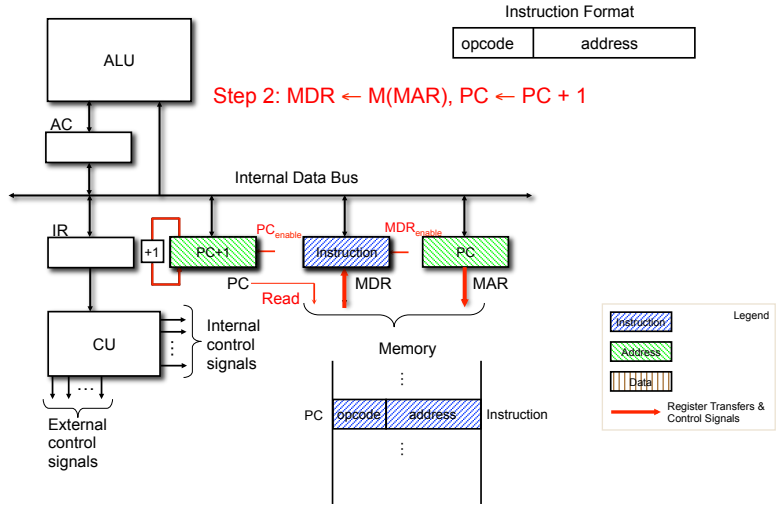
Go to beginning of Execute cycle

Note: Steps 2 and 4 can be performed at the same time.

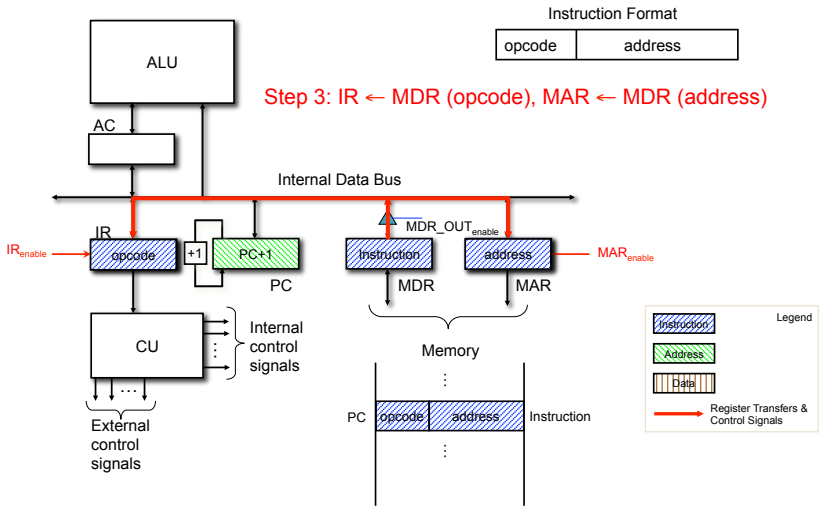
Fetch Cycle (Step 1)



Fetch Cycle (Step 2)



Fetch Cycle (Step 3)



Execute Cycle

- Execute cycle depends on the instruction
- Will describe execute cycle based on the following basic instruction:
  - Data transfer Instructions
    - LDA x (*Load Accumulator*)
    - STA x (*Store Accumulator*)
  - Arithmetic and Logical Instructions
    - ADD x (*Add to accumulator*)
  - Control Transfer
    - J x (*Jump to x*)
    - BNE x (*Branch conditionally to x*)

Execute Cycle

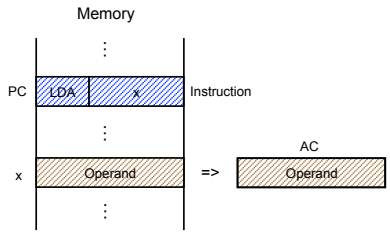
Example: LDA x (*Load Accumulator*)

Execute Cycle

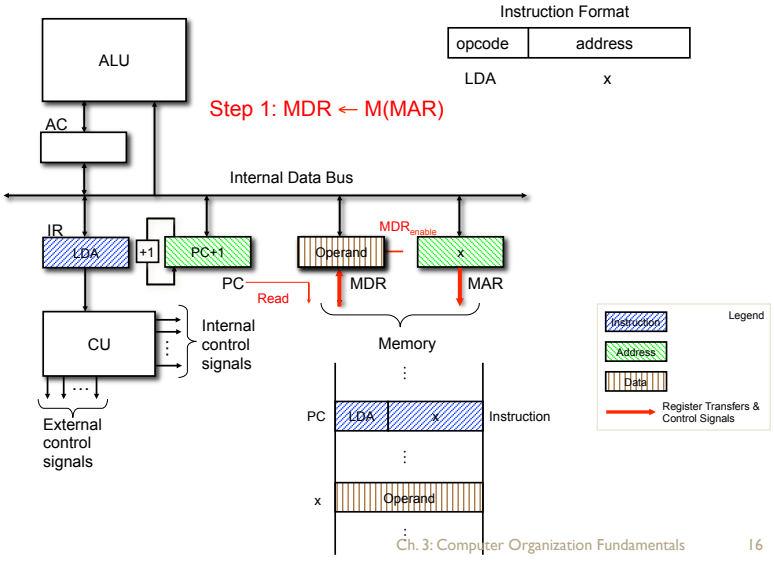
Step 1: MDR ← M(MAR)

Step 2: AC ← MDR

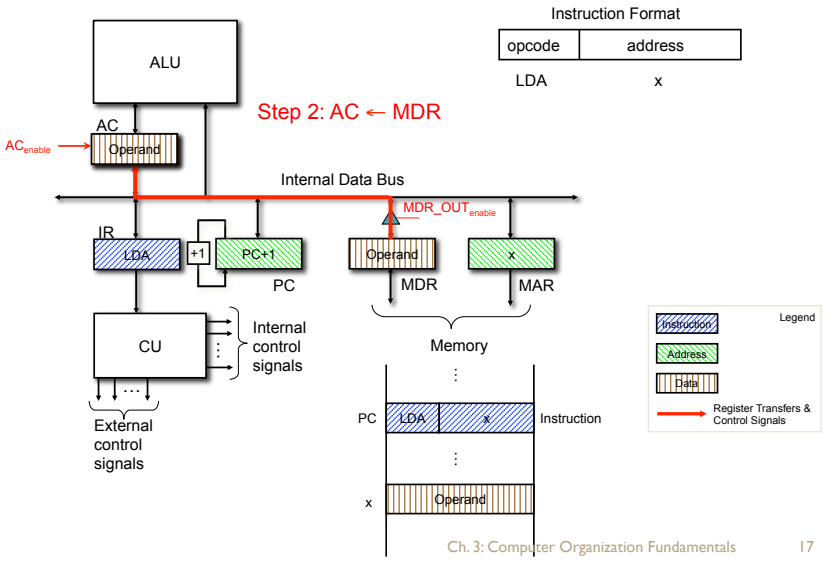
Return to the beginning of the instruction cycle



Execute Cycle (Step 1)



Execute Cycle (Step 2)



Execute Cycle

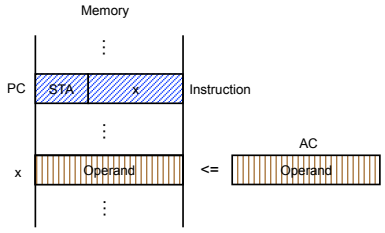
Example: STA x (*Store Accumulator*)

Execute Cycle

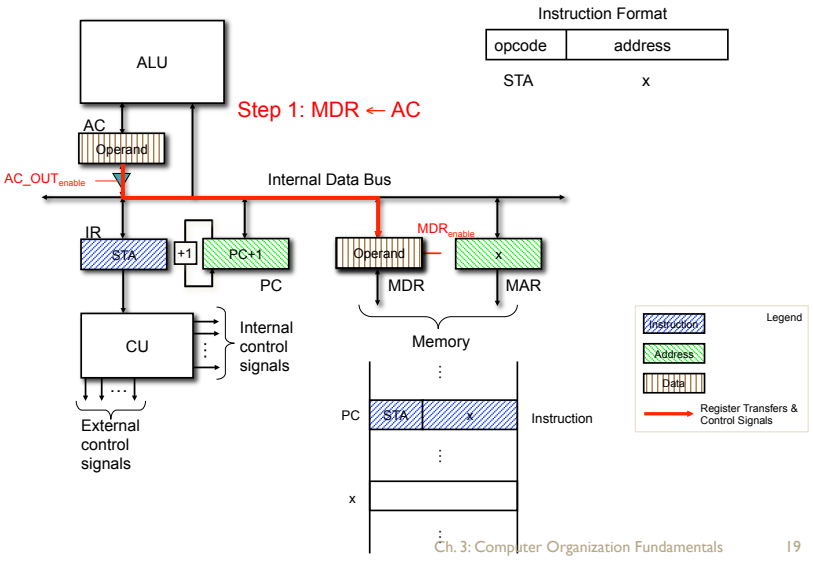
Step 1: MDR ← AC

Step 2: M(MAR) ← MDR

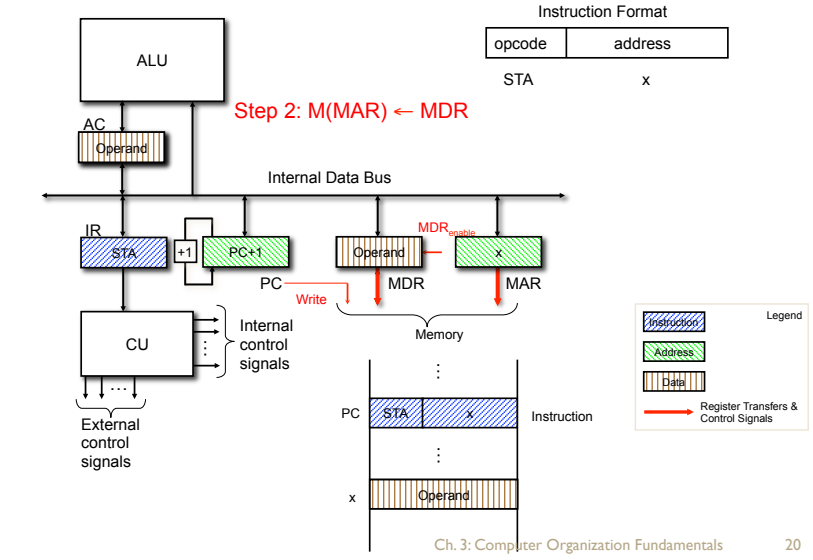
Return to the beginning of the instruction cycle



Execute Cycle (Step 1)



Execute Cycle (Step 2)



Execute Cycle

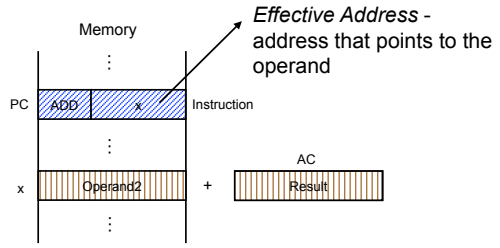
Example: ADD x (*Add to Accumulator*)

Execute Cycle

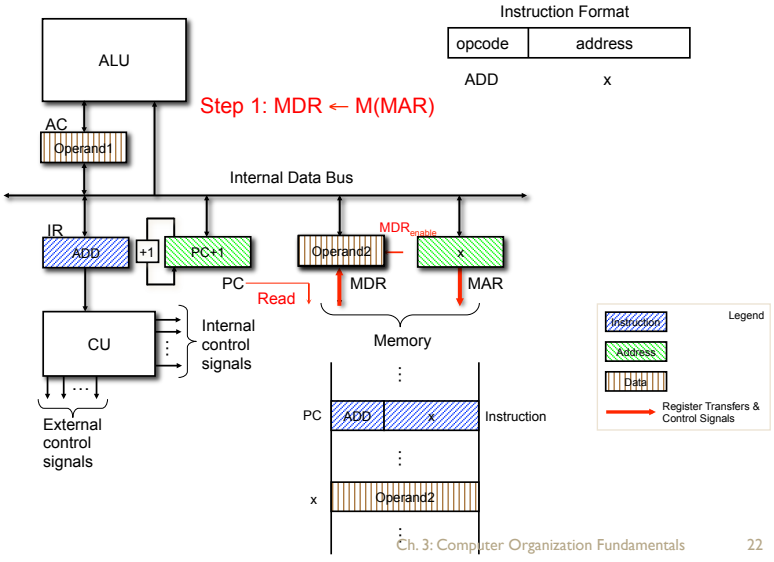
Step 1: MDR ← M(MAR) ; Read operand

Step 2: AC ← AC + MDR ; Add and transfer result to AC

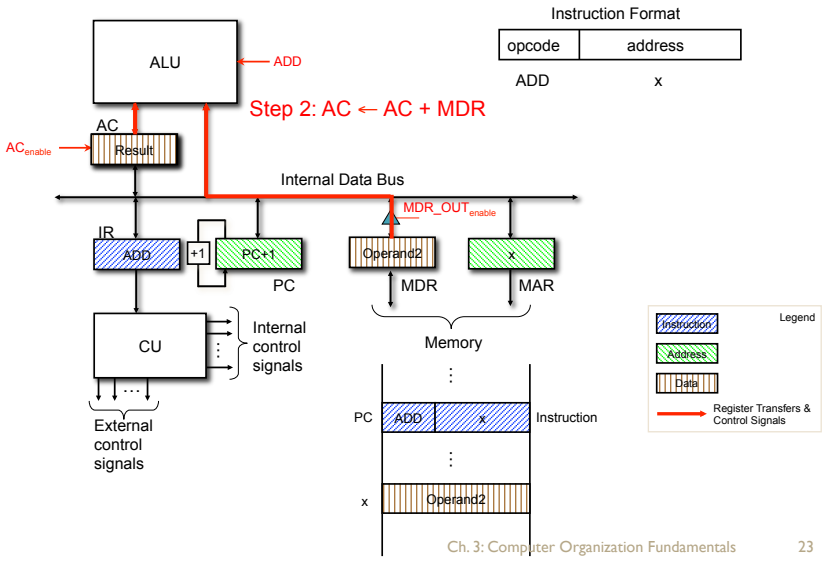
Return to the beginning of the instruction cycle



Execute Cycle (Step 1)



Execute Cycle (Step 2)



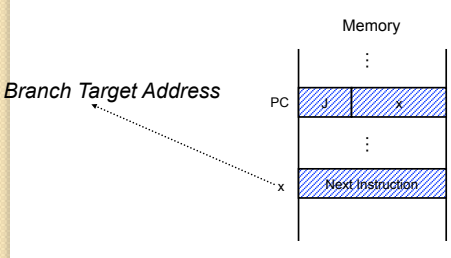
Execute Cycle

Example: J x (*Jump to x*)

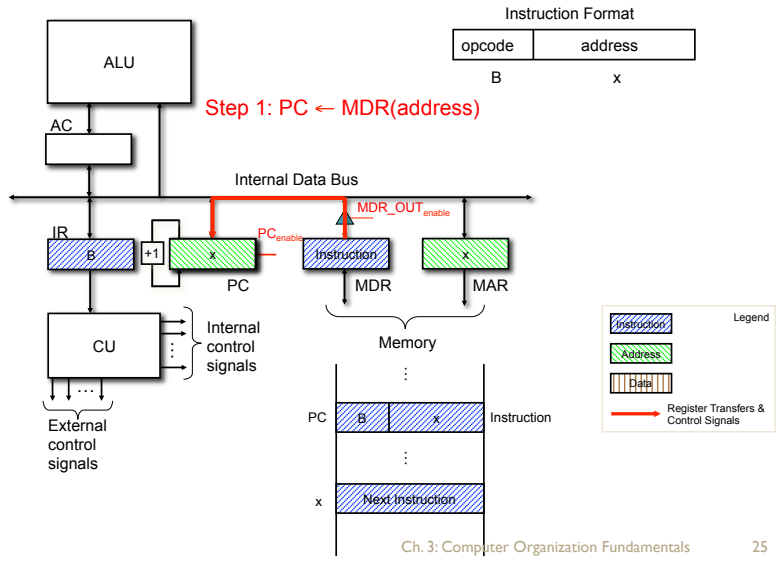
Execute Cycle

Step 1: PC ← MDR(address)

Return to the beginning of the instruction cycle



Execute Cycle (Step 1)



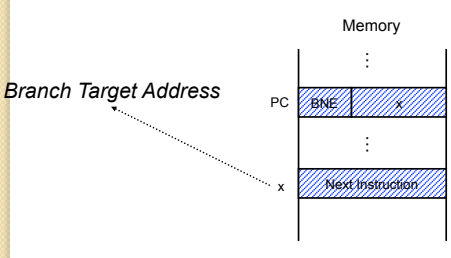
Execute Cycle

Example: BNE x (*Branch Conditionally to x*)

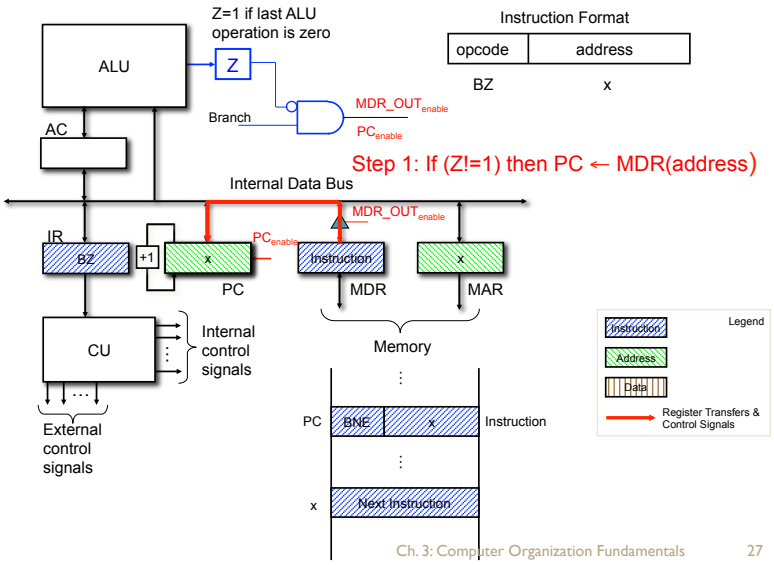
Execute Cycle

Step 1: If (Z!=1) then PC ← MDR(address)

Return to the beginning of the instruction cycle



Execute Cycle (Step 1)



One More Example…

Example: LDA (x) (*Load Accumulator Indirect*)

Execute Cycle

Step 1: MDR ← M(MAR); Read effective address (EA)

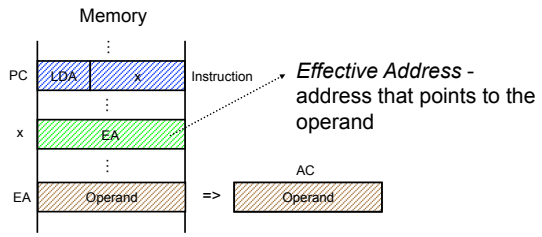
Step 2: MAR ← MDR ;

Step 3: MDR ← M(MAR) ; Read operand

Step 4: AC ← MDR ; Move operand to AC

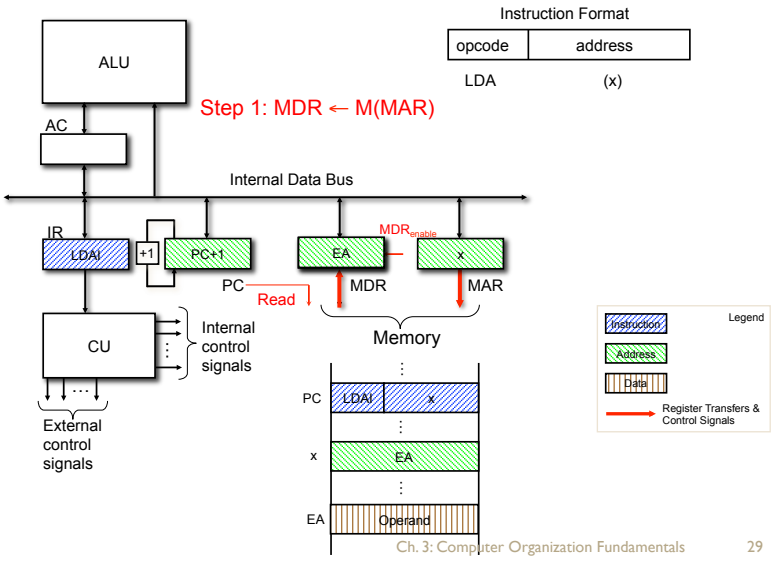
Return to the beginning of the instruction cycle

Useful for indexing arrays!

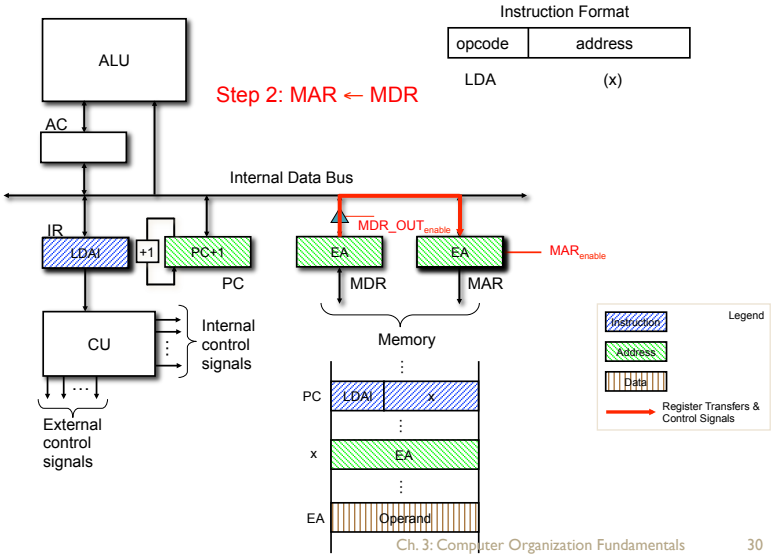


Effective Address - address that points to the operand

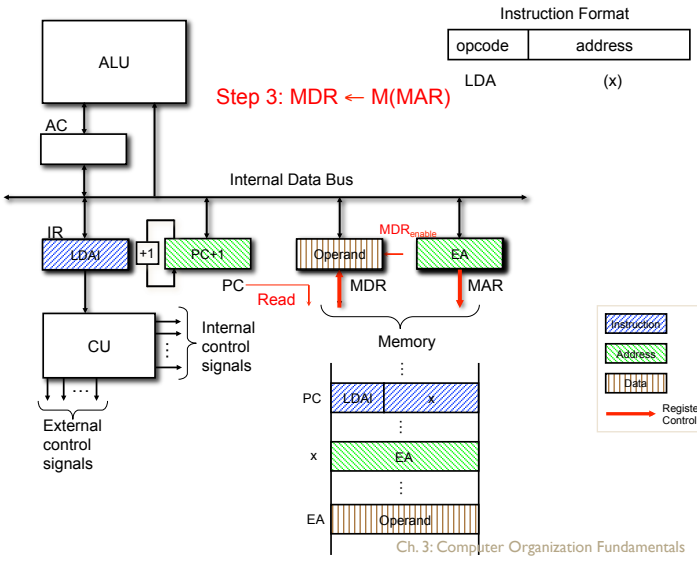
Execute Cycle (Step 1)



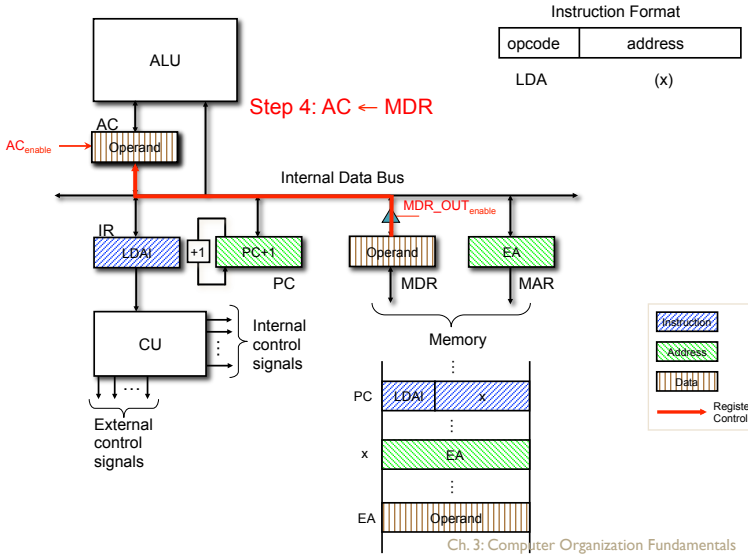
Execute Cycle (Step 2)



Execute Cycle (Step 3)



Execute Cycle (Step 4)



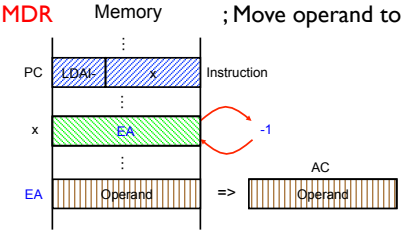
Last Example…(I promise!)

Example: LDA -(x) (*Load Accumulator Indirect with Pre-decrement*)

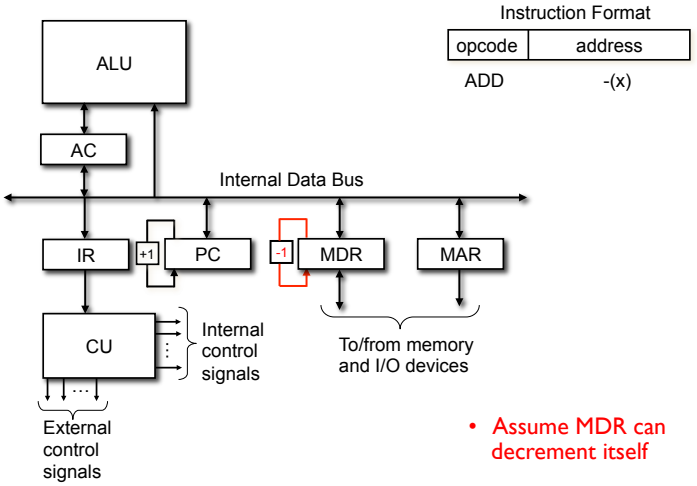
Useful for stepping through arrays

Execute Cycle

Step 1:	MDR ← M(MAR)	; Read effective address (EA)
Step 2:	MDR ← MDR - 1	; Decrement EA
Step 3:	M(MAR) ← MDR	; Store it back in x
Step 4:	MAR ← MDR	;
Step 5:	MDR ← M(MAR)	; Read operand
Step 6:	AC ← MDR	; Move operand to AC

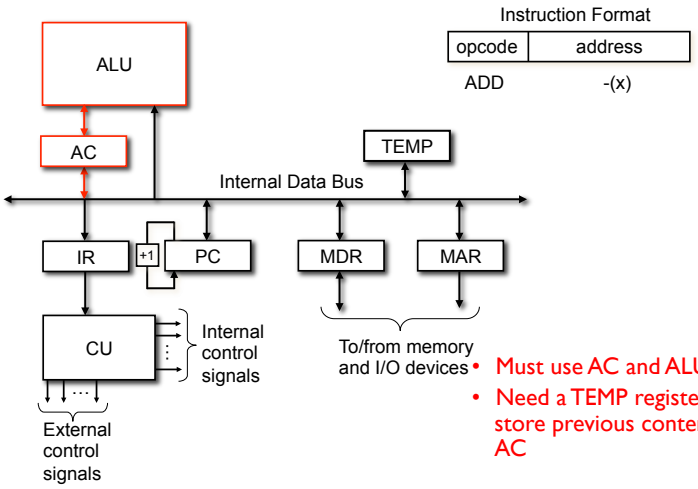


Easiest Way



- Assume MDR can decrement itself

Hard Way



- Must use AC and ALU!
- Need a TEMP register to store previous content of AC

Hard Way

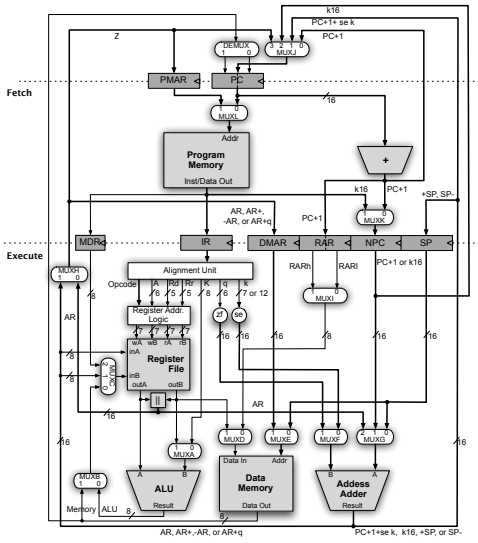
- MDR doe not have the capability to decrement itself.
- So must use ALU through AC.
- AC needs to be saved so that it is not overwritten.

Execute Cycle

Step 1:	MDR ← M(MAR)	; Read effective address (EA)
Step 2:	Temp ← AC	; Save AC in Temp
Step 3:	AC ← MDR	;
Step 4:	AC ← AC - 1	; Decrement EA
Step 5:	MDR ← AC	
Step 6:	AC ← Temp	; Restore AC
Step 7:	M(MAR) ← MDR	; Store it back in x
Step 8:	MAR ← MDR	;
Step 9:	MDR ← M(MAR)	; Read operand
Step 10:	AC ← MDR	; Move operand to AC

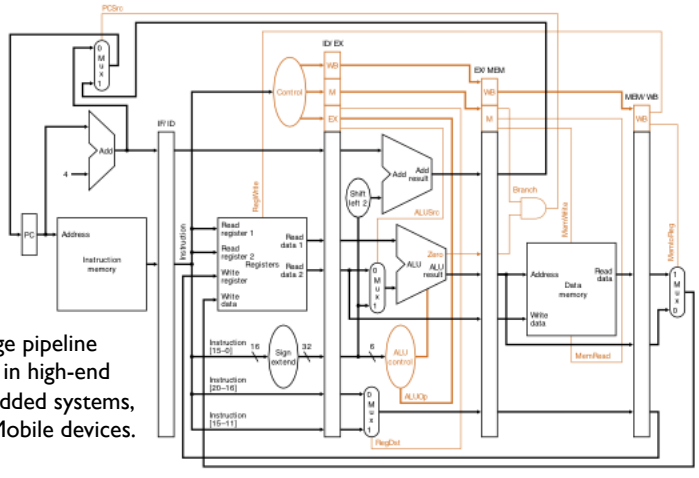
- Can you think of a way to perform LDA (x)+ (*Load Accumulator Indirect with Post-increment*)?

What We Will See Later…



- AVR Microcontroller
- Used in low-end embedded systems

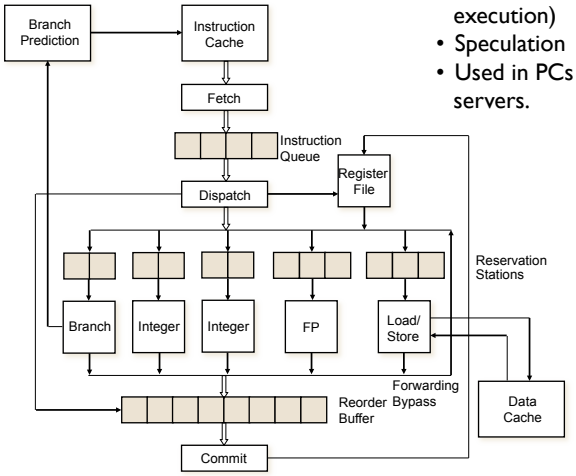
What You Will See in ECE 472…



- 5-stage pipeline
- Used in high-end embedded systems, e.g., Mobile devices.

What You Will See in ECE 570…

- SuperScalar
- OoO (out-of-order execution)
- Speculation
- Used in PCs and servers.



Questions?