Chapter 3: Computer Organization Fundamentals

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

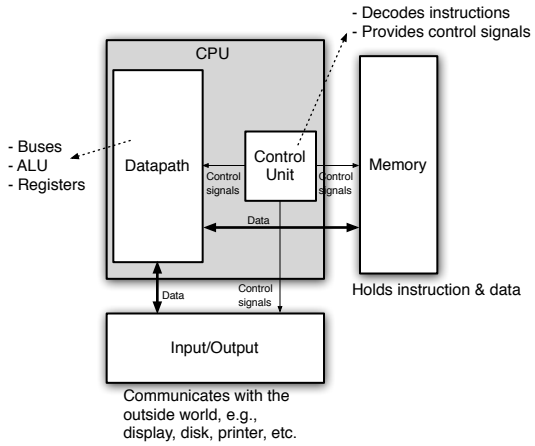
1

Chapter Goals

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

2

Computer Organization



3

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

4

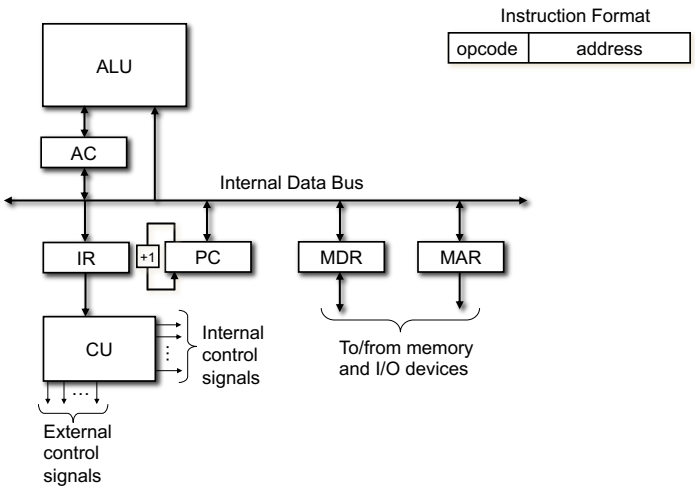
Registers

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).

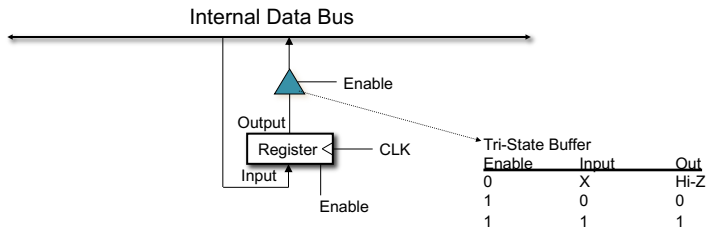
5

A Pseudo-CPU



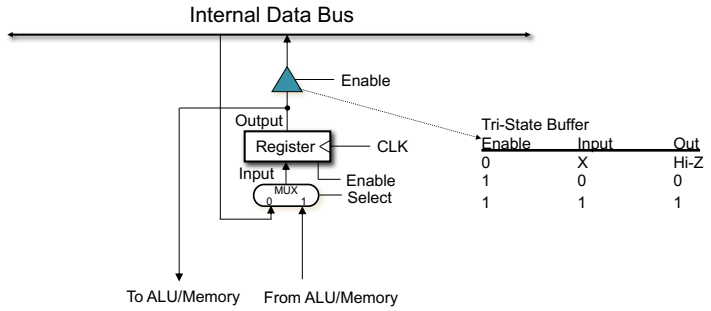
6

Bus-Register Connections



7

Bus-Register Connections



8

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

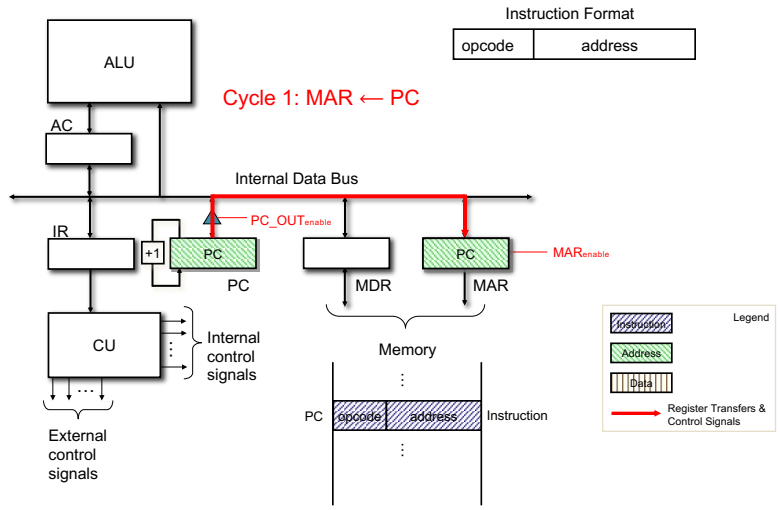
- Need to describe what has to happen in each cycle.
- Will use register transfer operations to describe the movement of data.

Fetch Cycle

Cycle 1: MAR ← PC
Cycle 2: MDR ← M[MAR] ; Read the content of memory
; location pointed to by MAR
Cycle 3: IR ← MDR(opcode), MAR ← MDR(address)
Cycle 4: PC ← PC + 1
Go to beginning of Execute cycle

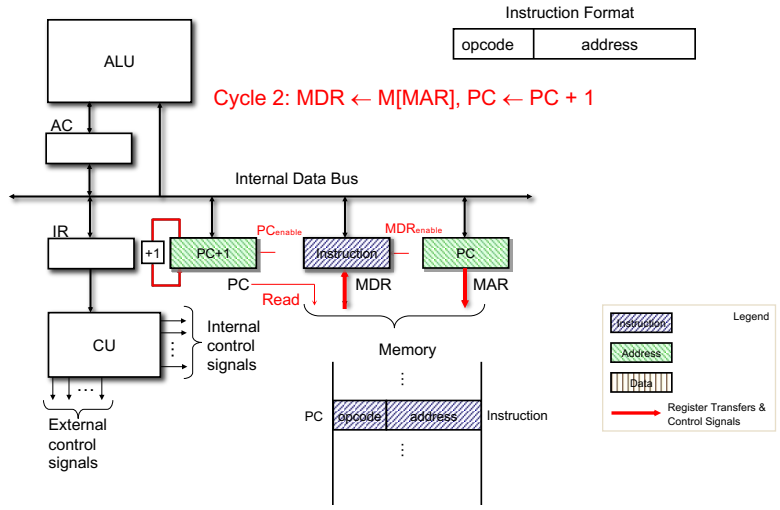
Note: Cycles 2 and 4 can be performed at the same time.

Fetch Cycle (Step 1)



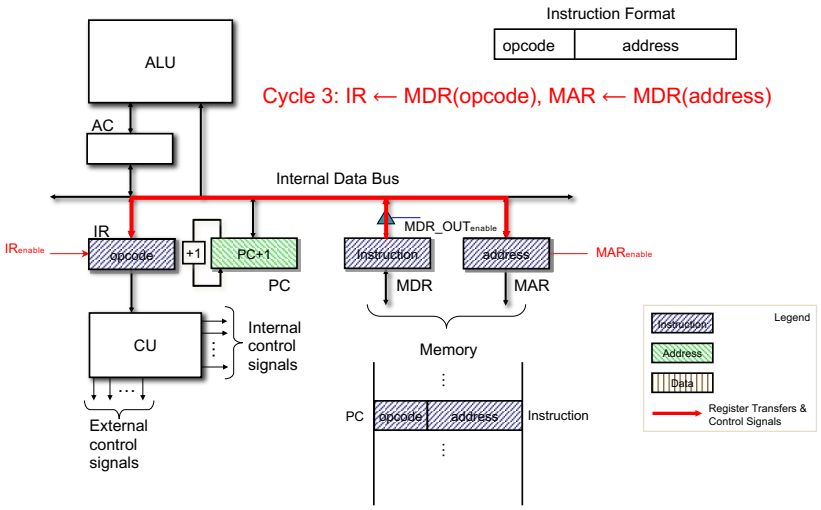
11

Fetch Cycle (Step 2)



12

Fetch Cycle (Step 3)



13

Execute Cycle

- Execute cycle depends on the instruction
- Will describe execute cycle based on instruction in the pseudo-ISA:
  - Data transfer Instructions
    - LDA x (*Load Accumulator*) ✓
    - STA x (*Store Accumulator*) ✓
  - Arithmetic and Logical Instructions
    - ADD x (*Add to accumulator*) ✓
    - SUB x (*Subtract from accumulator*)
    - NAND x (*Logical NAND to accumulator*)
    - SHFT (*Shift accumulator*)
  - Control Transfer
    - J x (*Jump to x*) ✓
    - BNE x (*Branch conditionally to x*) ✓

14

Execute Cycle

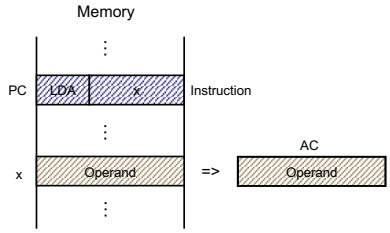
Example: LDA x (*Load Accumulator*)

Execute Cycle

Cycle 1: MDR ⟵ M[MAR]

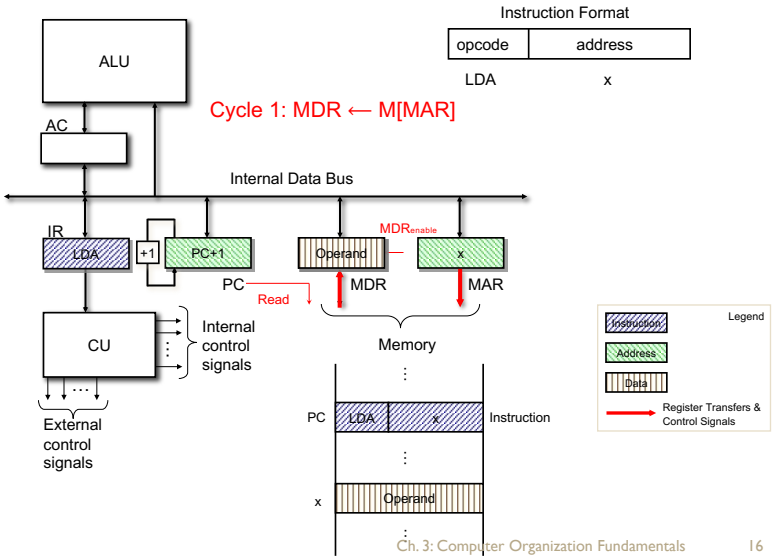
Cycle 2: AC ⟵ MDR

Return to the beginning of the instruction cycle



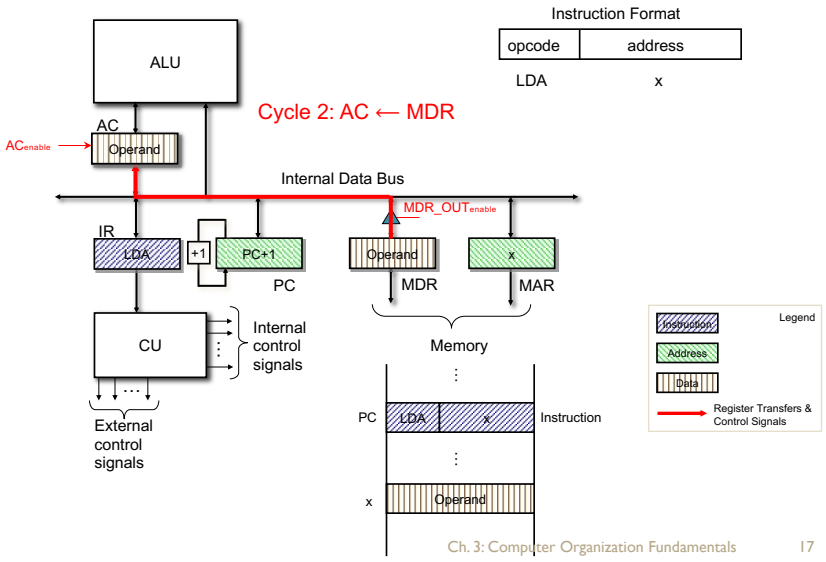
15

Execute Cycle (Step 1)



16

Execute Cycle (Step 2)



17

Execute Cycle

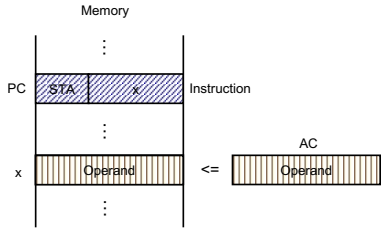
Example: STA x (*Store Accumulator*)

Execute Cycle

Cycle 1: MDR ⟵ AC

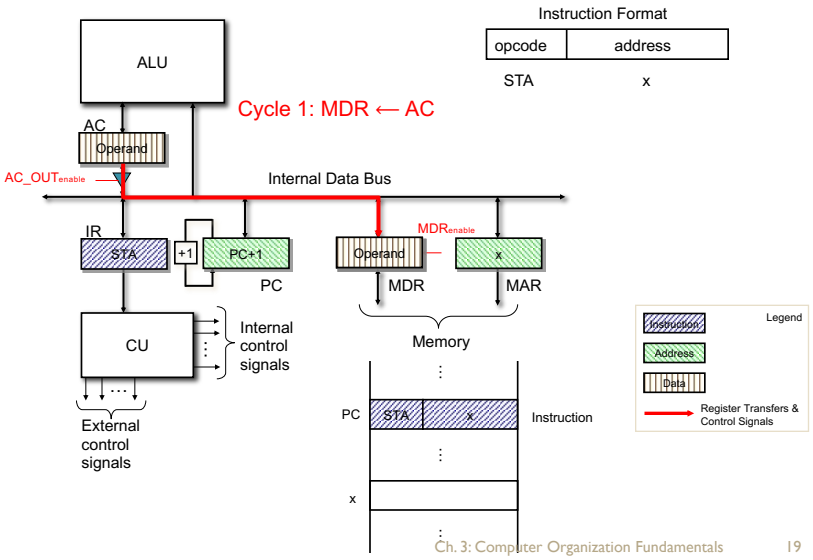
Cycle 2: M[MAR] ⟵ MDR

Return to the beginning of the instruction cycle



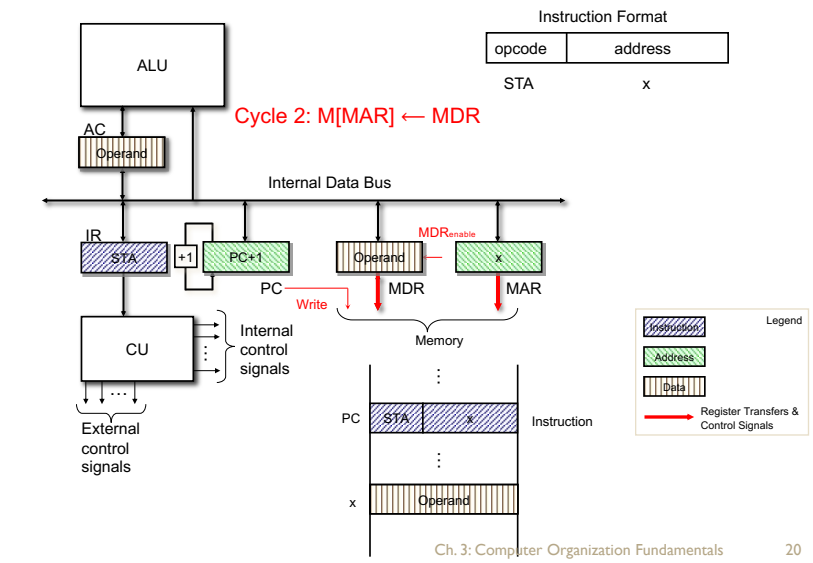
18

Execute Cycle (Step 1)



19

Execute Cycle (Step 2)



20

Execute Cycle

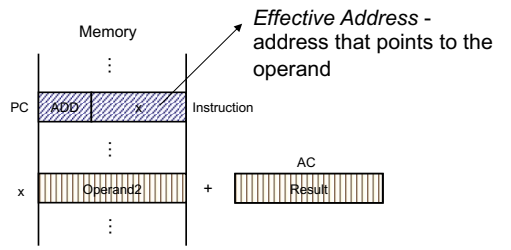
Example: ADD x (*Add to Accumulator*)

Execute Cycle

Cycle 1: MDR ⟵ M[MAR] ; Read operand

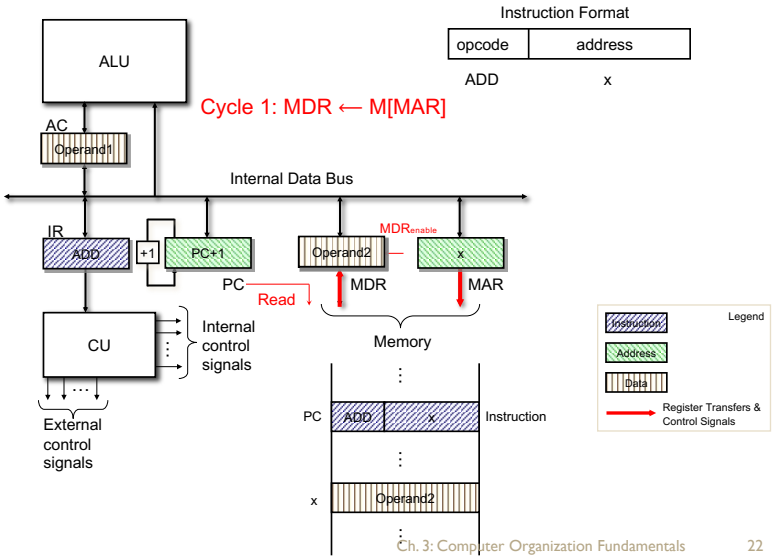
Cycle 2: AC ⟵ AC + MDR ; Add and transfer result to AC

Return to the beginning of the instruction cycle



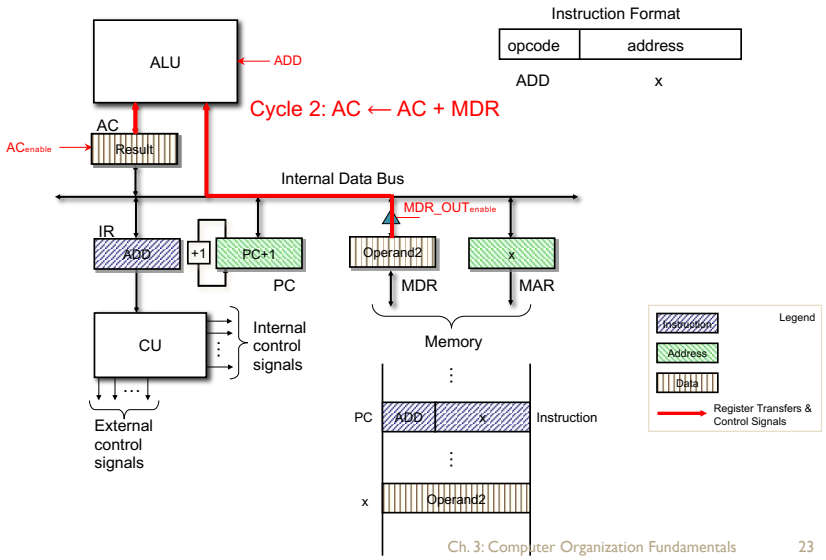
21

Execute Cycle (Step 1)



22

Execute Cycle (Step 2)



23

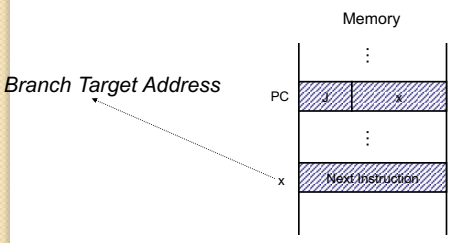
Execute Cycle

Example: J x (*Jump to x*)

Execute Cycle

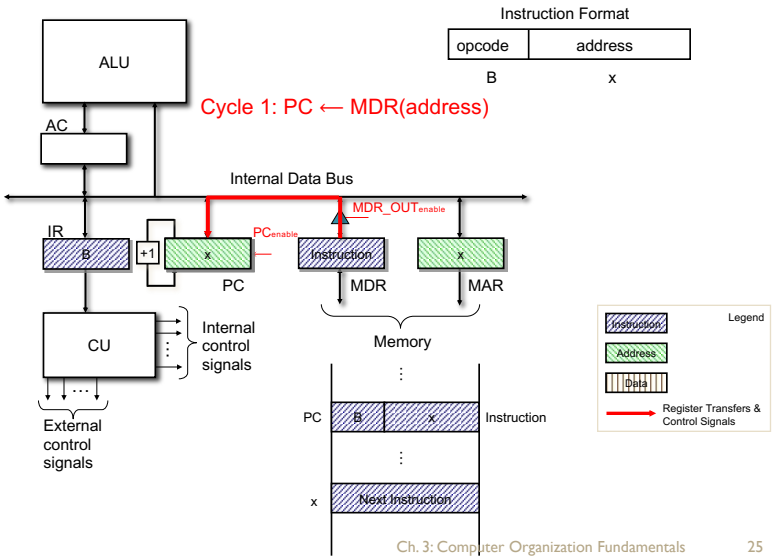
Cycle 1: PC ⟵ MDR(address)

Return to the beginning of the instruction cycle



24

Execute Cycle (Step 1)



25

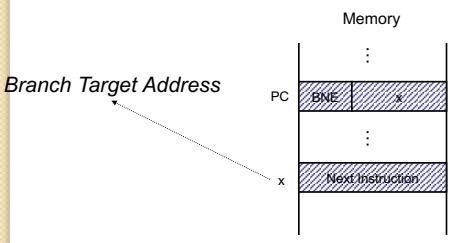
Execute Cycle

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

Execute Cycle

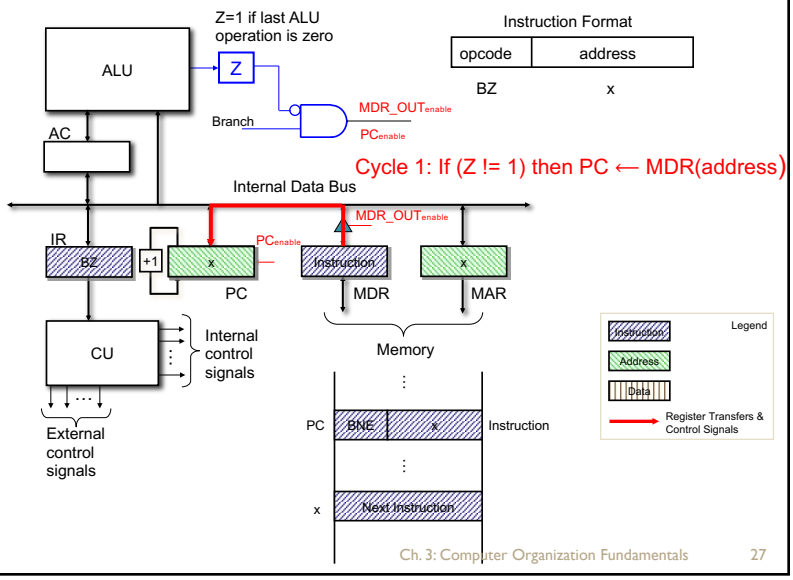
Cycle 1: If (Z != 1) then PC ⟵ MDR(address)

Return to the beginning of the instruction cycle



26

Execute Cycle (Step 1)



27

One More Example…

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

Execute Cycle

Cycle 1: MDR ⟵ M[MAR] ; Read effective address (EA)

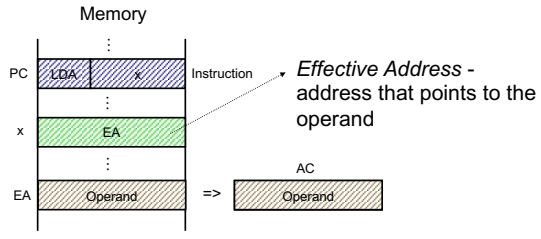
Cycle 2: MAR ⟵ MDR ; Move EA to MAR

Cycle 3: MDR ⟵ M[MAR] ; Read operand

Cycle 4: AC ⟵ MDR ; Move operand to AC

Return to the beginning of the instruction cycle

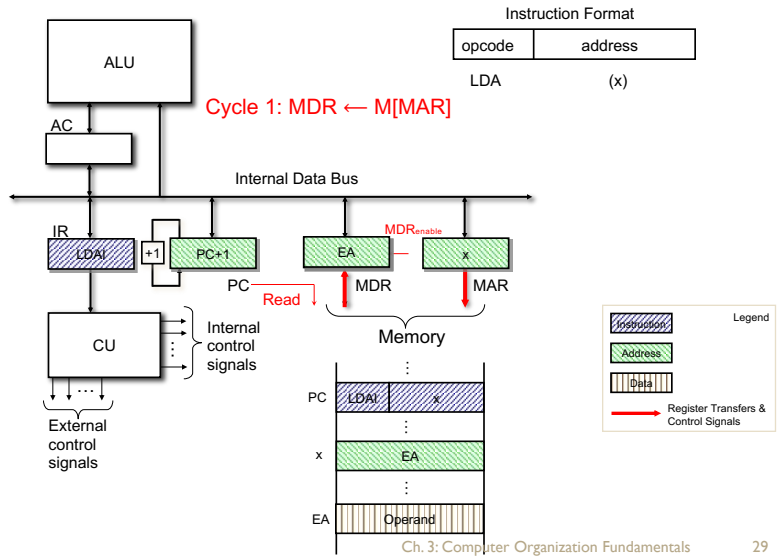
Useful for indexing arrays!



Effective Address - address that points to the operand

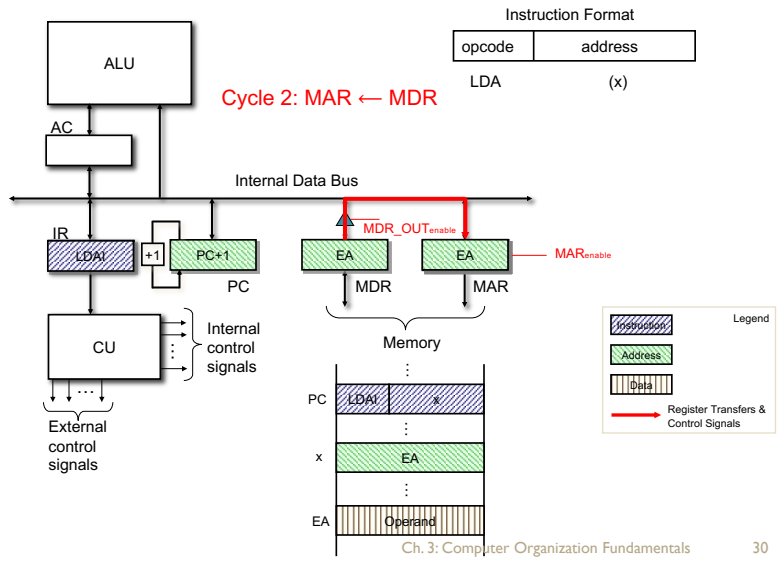
28

Execute Cycle (Step 1)



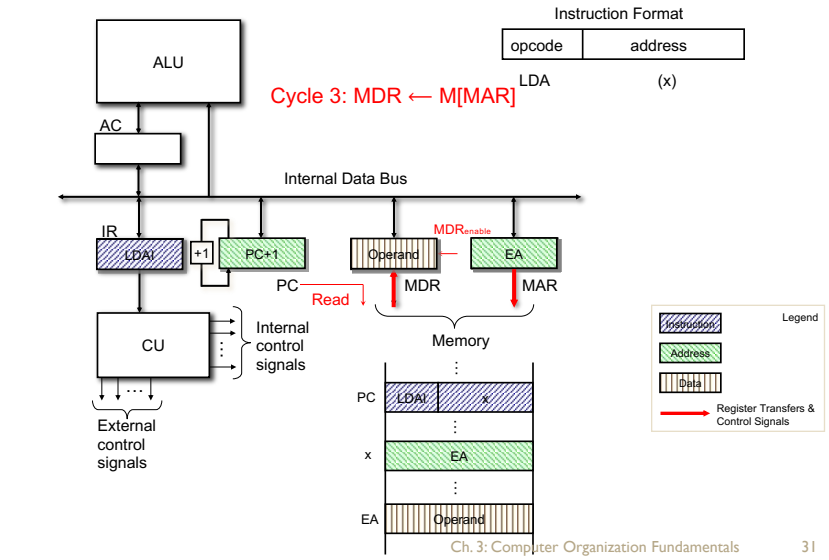
29

Execute Cycle (Step 2)



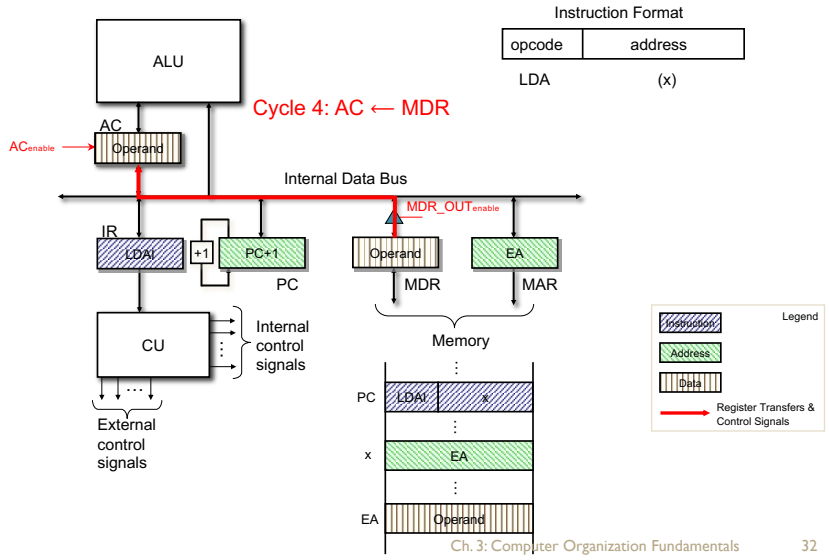
30

Execute Cycle (Step 3)



31

Execute Cycle (Step 4)



32

Last Example…(I promise!)

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

Useful for stepping through arrays

Execute Cycle

Cycle 1: MDR ← M[MAR] ; Read EA+1

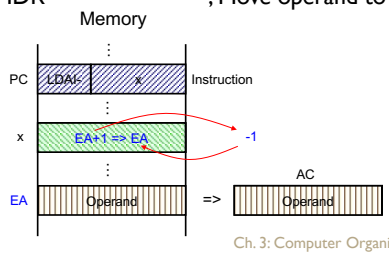
Cycle 2: MDR ← MDR - 1 ; Decrement EA

Cycle 3: M[MAR] ← MDR ; Store it back in location x (i.e., M[x])

Cycle 4: MAR ← MDR ; Move EA to MAR

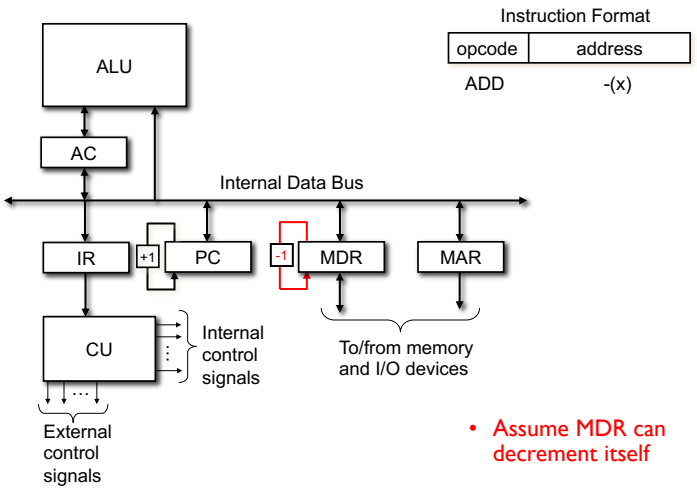
Cycle 5: MDR ← M[MAR] ; Read operand

Cycle 6: AC ← MDR ; Move operand to AC



33

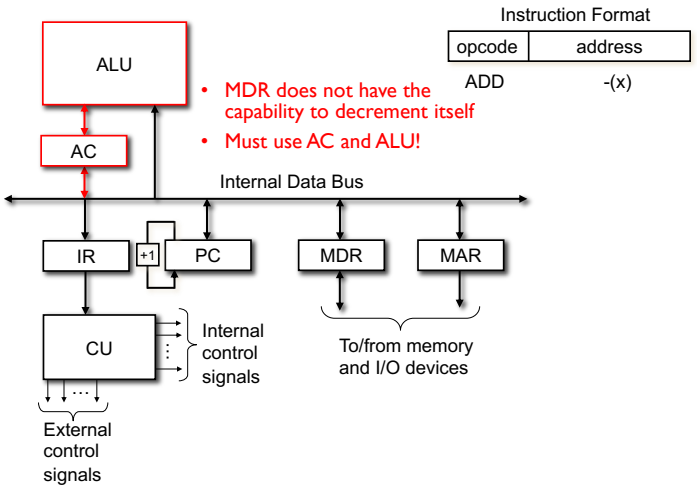
Easiest Way



- Assume MDR can decrement itself

34

Hard Way (1)



- MDR does not have the capability to decrement itself
- Must use AC and ALU!

35

Hard Way (2)

Execute Cycle

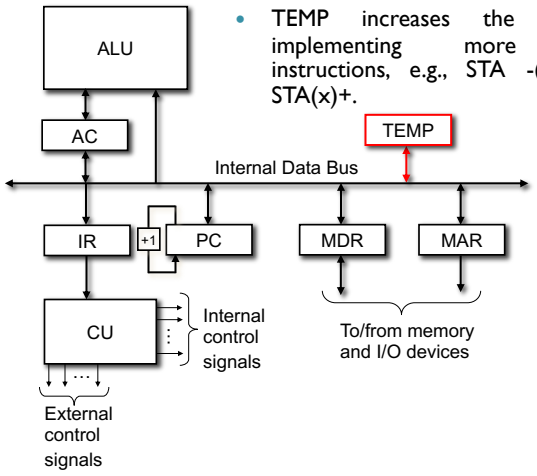
Cycle 1: $MDR \leftarrow M[MAR]$; Read EA+1
Cycle 2: $AC \leftarrow MDR$; Move EA+1 to AC
Cycle 3: $AC \leftarrow AC - 1$; Decrement EA+1
Cycle 4: $MDR \leftarrow AC$; Move EA to MDR
Cycle 5: $M[MAR] \leftarrow MDR$; Store it back in location x (i.e., M[x])
Cycle 6: $MAR \leftarrow MDR$; Move EA MAR
Cycle 7: $MDR \leftarrow M[MAR]$; Read operand
Cycle 8: $AC \leftarrow MDR$; Move operand to AC

Note: Cycles 5 and 6 can be performed at the same time.

36

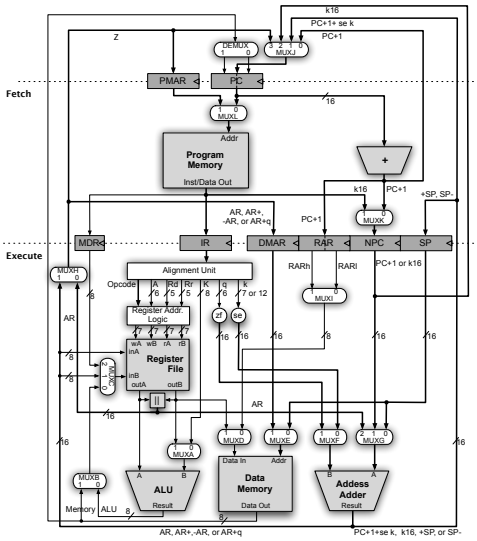
Pseudo-CPU with TEMP Register

- There may be times when the content of AC has to be preserved.
- TEMP increases the flexibility for implementing more complicated instructions, e.g., STA -(x), LDA (x)+, STA(x)+.



37

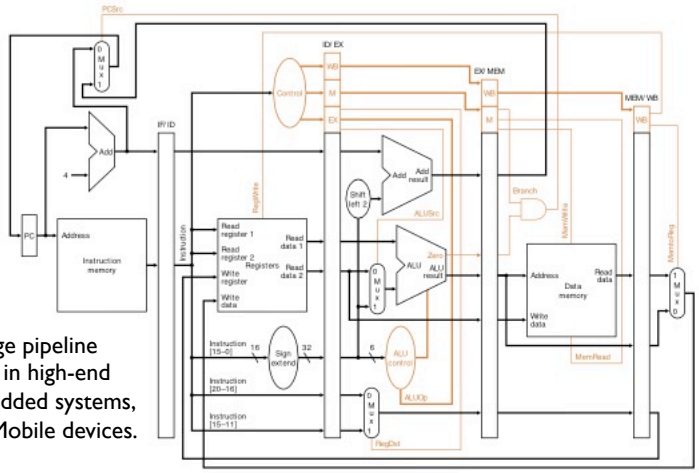
What We Will See Later…



- AVR Microcontroller
- Used in low-end embedded systems

38

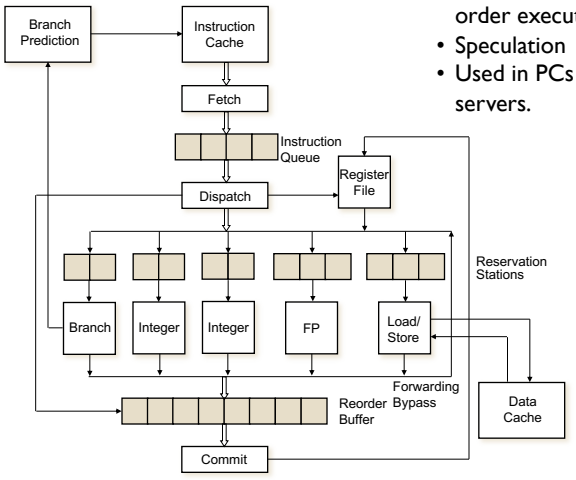
What You Will See in ECE 472…



- 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?