CEG3136 Computer Architecture II

Module 1 - Introduction

uOttawa

L'Université canadienne
Canada's university

Notes from

Dr. Voicu Groza

Topics of discussion

- Course syllabus
- Virtual Campus
- Course Organization
- Introduction – the picoprocessor
- Reading: Chapter 1 and Sections 2.1-2.2

Unless otherwise noted, sources for the figures in the course notes are given in the last slide

Course Syllabus

- Provides:
  - Information on Instructor and TAs
  - Lecture, tutorial, lab schedules
  - Course Description and prerequisites
  - Course Objectives
  - Text and References
  - Grading
  - Information on labs
  - Course Outline
  - Available on Virtual Campus

Course Virtual Campus site

- Access site: **uottawa.brightspace.com**
  - Log in and choose the course CEG3136
  - Use your InfoWeb user name and password to access the Virtual Campus
- You will have access to:
  - Course syllabus;
  - All course material;
  - Labs;
  - Assignments tool that allows you to submit lab reports electronically;
  - Grading;
  - Announcement tool.

CEG 3136 Syllabus

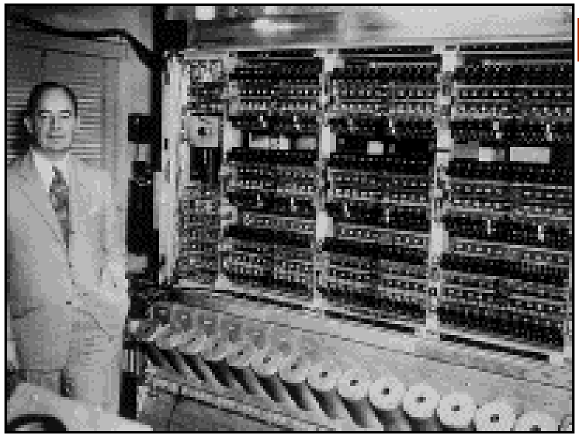
- Class attendance is mandatory. As per academic regulations, students who do not attend 80% of the class will not be allowed to write the final examinations.

- All components of the course (i.e. laboratory repots, assignments, etc.) must be fulfilled; otherwise students may receive an INC as a final mark (equivalent to F).

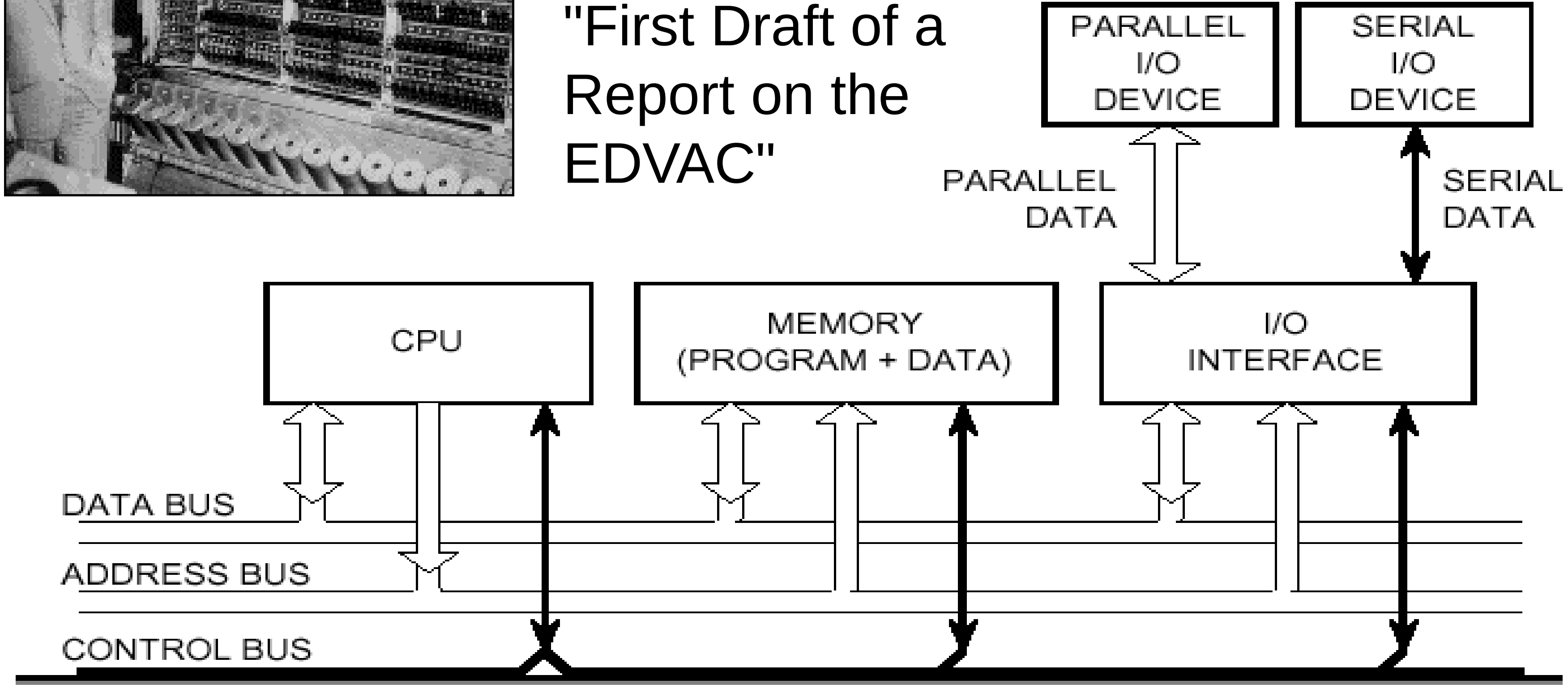
Introduction: the picoprocessor

- Objective
  - Simple design of a processor
  - Executes a few simple instructions
  - Gain understanding of processor hardware principles
- Basic idea
  - Processor is basically a collection of digital logic components

Basic Computer Architecture



- Based on von Neumann architecture
  - Developed in 1945
  - "First Draft of a Report on the EDVAC"



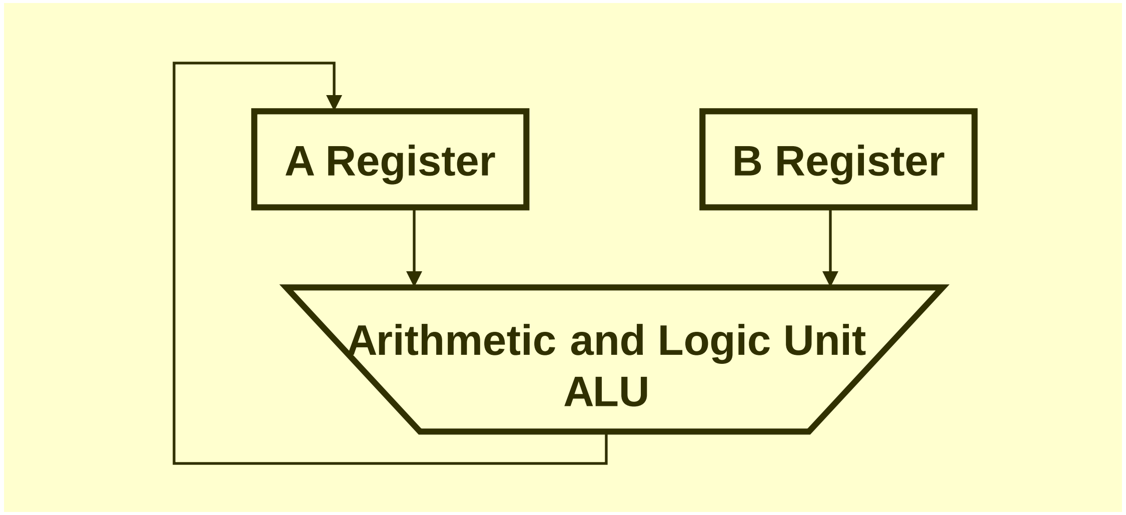
Computer Operations and Op codes

- What are the processor operations?
- Need to represent with binary codes for the processor.
- Mnemonic is a human readable representation of the operation
  - Is the basis for creating an assembler language

Operation (mnemonic)	Operation Code
ADD	00
SUB	01
IN	10
OUT	11

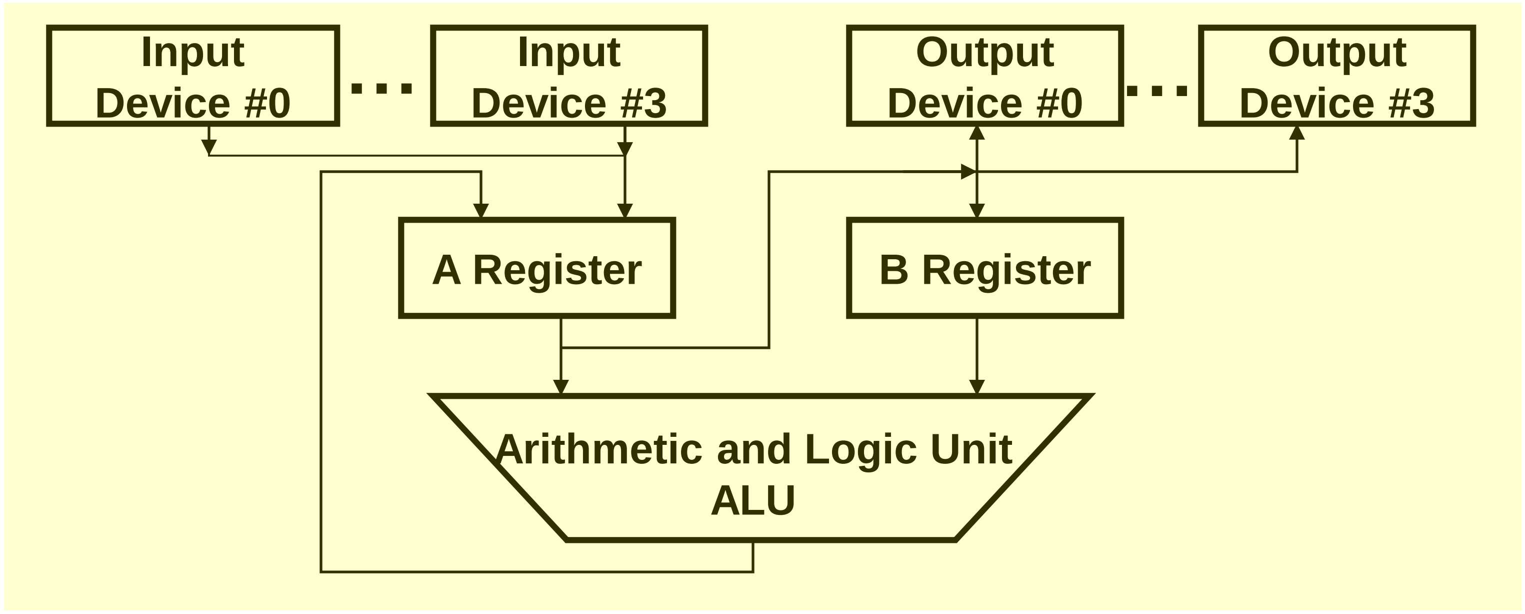
Data Path: Hardware for Addition and Subtraction

- Add and subtract two binary numbers
- Numbers held in registers
- ALU contains ripple-carry full-adder for adding
  - Similar hardware for subtraction
- A register provides one operand but also receives the result - accumulator



Data Path: Adding Input-Output Devices

- Need *source* (input) device to get operands
  - Example: Set of eight switches
- Need *destination* (output) device to display result
  - Example: Set of 8 LEDs



Adding operands to the IN and OUT operations

- Use ***operand*** with our IN and OUT instructions
  - Operand provides the device number (address)
- ***Operation code*** + ***operand*** gives processor ***instruction***
  - Use 8-bits to define a processor instruction (includes operands ii and oo)

Operation	Operand	Instruction Code
ADD	None	`00------`
SUB	None	`01------`
IN	Device # **`ii`**	`10----ii`
OUT	Device # oo	`11----oo`

The MOV instruction

- Required to move (actually **copy**) the contents from Accumulator A to Register B
  - Note, need to use another bit to define the operation

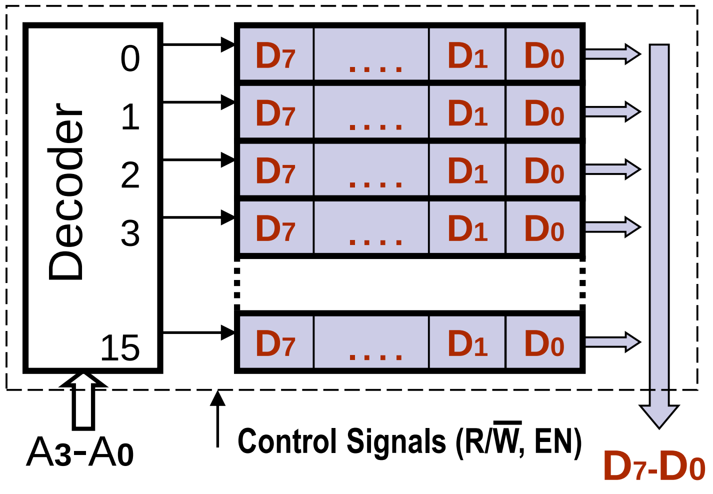
Operation	Operand	Instruction Code
ADD	None	`001-----`
SUB	None	`011-----`
IN	Device #	`101---ii`
OUT	Device #	`111---oo`
MOV	None	`010-----`

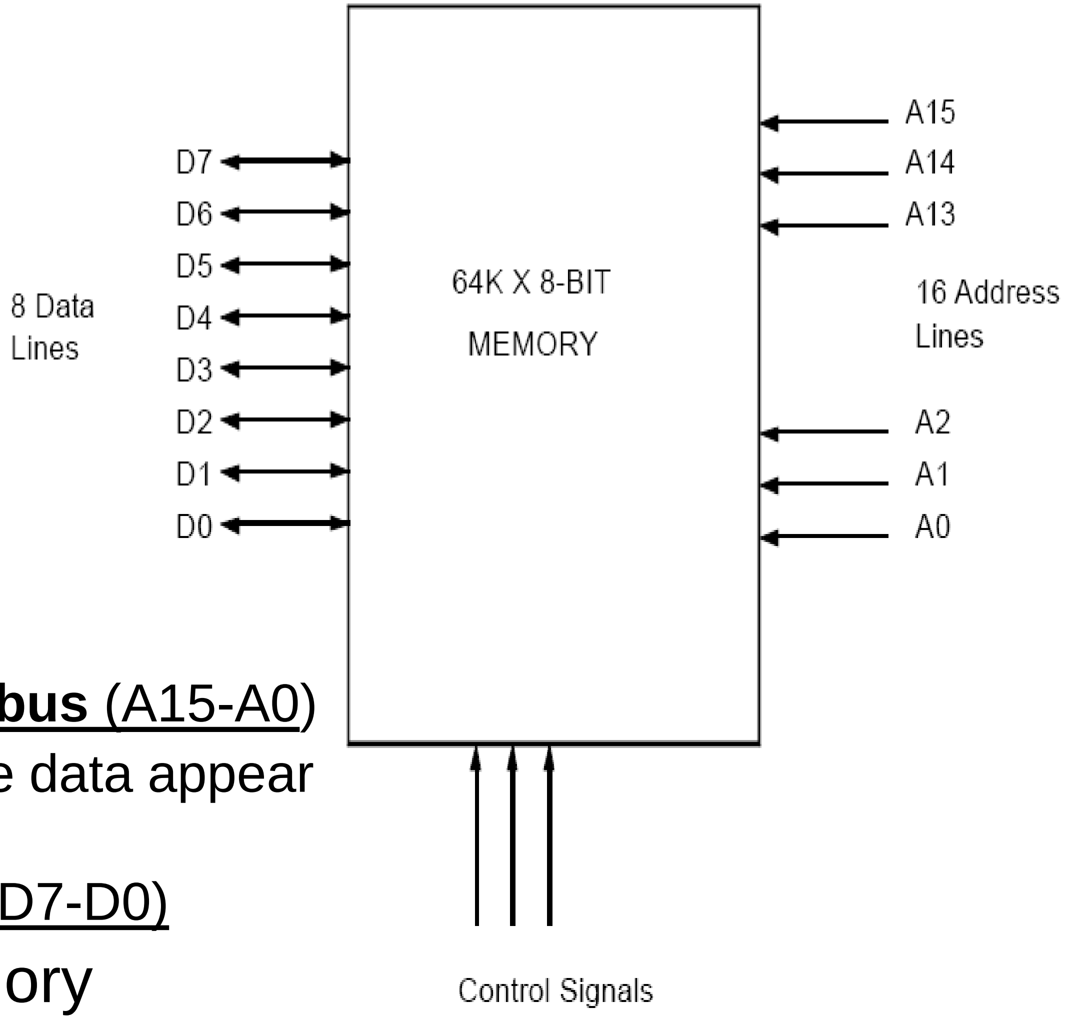
uOttawa

Adding Two Numbers

Line	Operation Field	Operand Field	Comment Field
1	IN	1	(A) ← (Switches)
2	MOV		(B) ← (A)
3	IN	1	(A) ← (Switches)
4	ADD		(A) ← (A) + (B)
5	OUT	2	(LED) ← (A)

Computer Memory





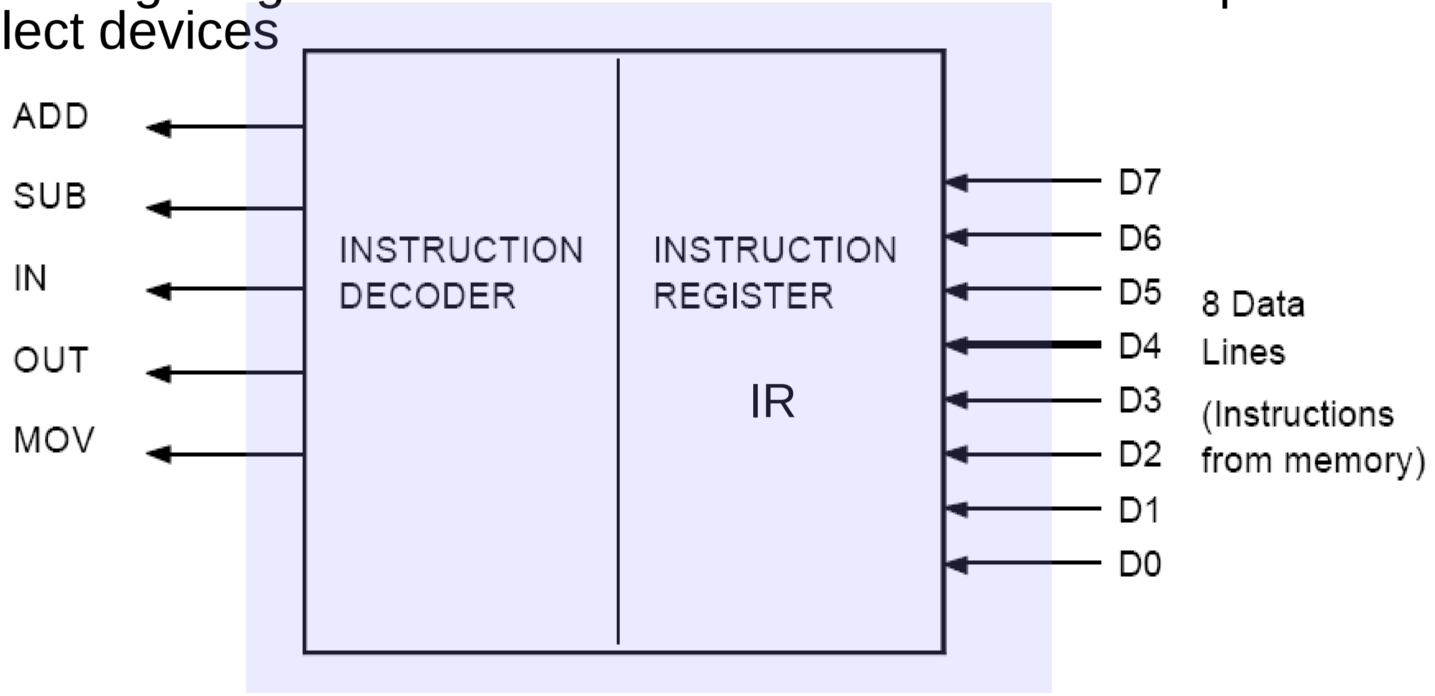
- Read operation
  - Apply address to **address bus** (A15-A0)
  - Apply control signal to have data appear on data bus (RD)
  - Read data from **data bus** (D7-D0)
- How to write data to memory location?

Program in Computer Memory

	Operation Field		Comment Field	Addr	**Contents**
1	IN	1	(A) ← (Switches)	0:	1010 0001
2	MOV		(B) ← (A)	1:	0100 0000
3	IN	1	(A) ← (Switches)	2:	1010 0001
4	ADD		(A) ← (A) + (B)	3:	0010 0000
5	OUT	2	(LED) ← (A)	4:	1110 0010

Instruction Register and Decoder

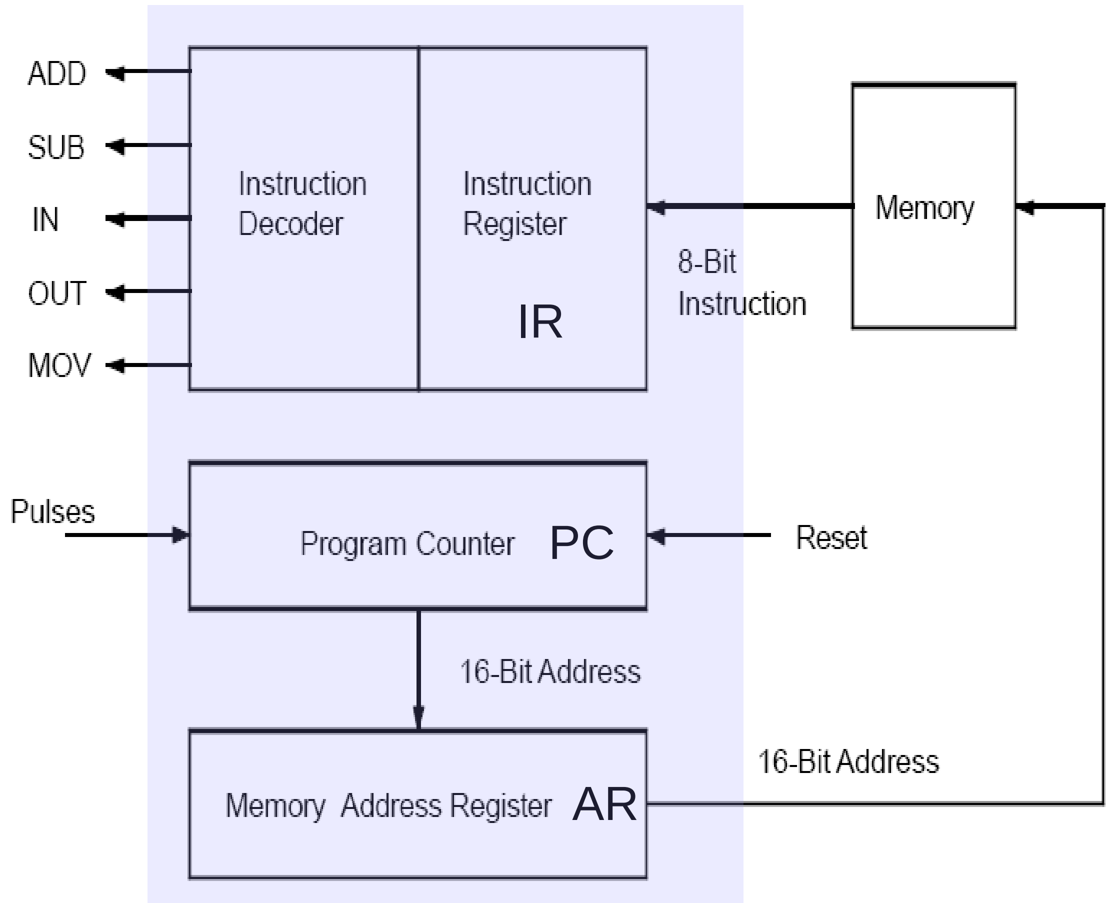
- Instruction register
  - Used to hold the computer instruction
- Instruction decoder
  - Logic circuit that decodes instruction
  - Asserts logic signals for each operation – example IN used as clock signal to latch data into the A register
  - Other logic signals not shown – decode OUT and IN operands to select devices



uOttawa

Program Counter and Memory Address Register

- To sequence through program



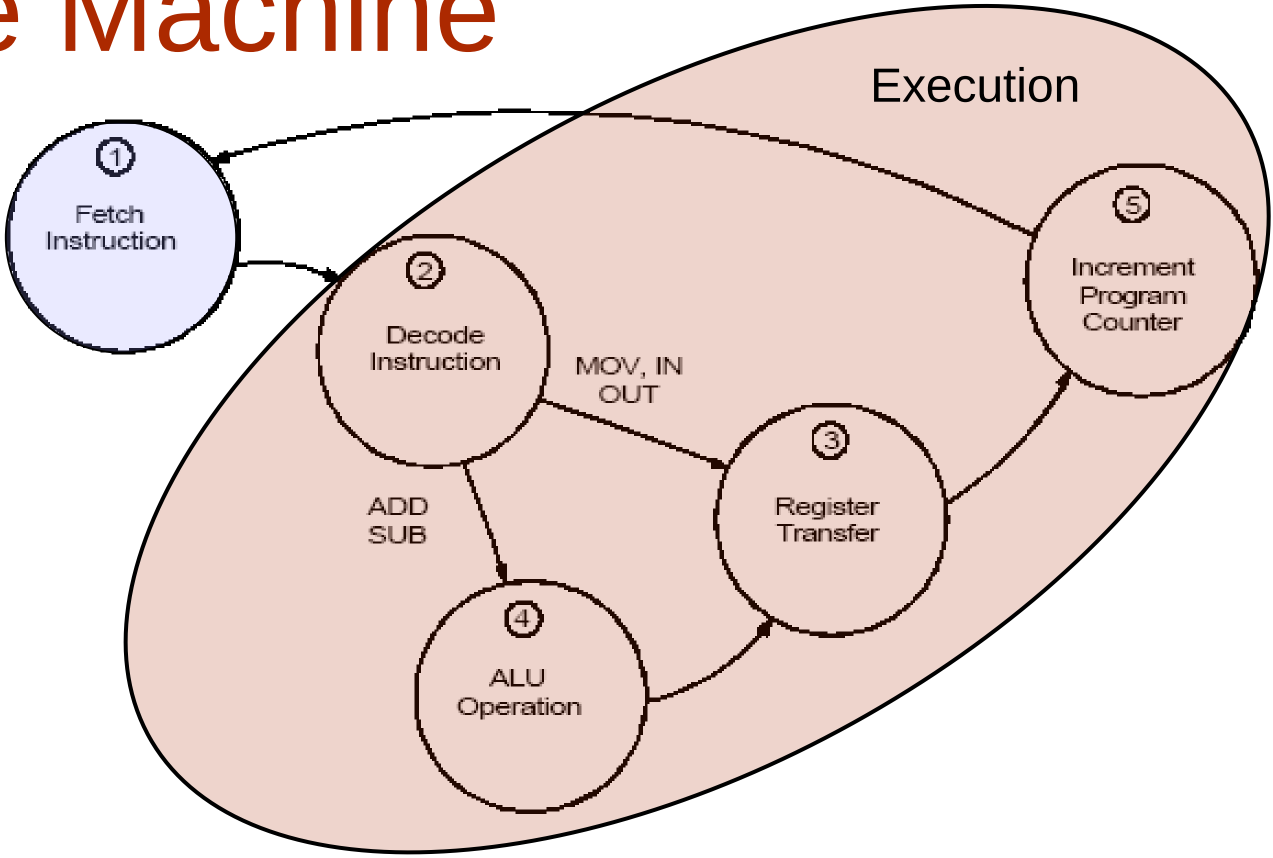
(IN) Instruction Execution Cycle

- Need to start thinking about time it takes to execute instructions
- Consider the steps to execute an instruction
  1. Fetch instruction
     - Transfer contents of PC to Memory Address Register
     - Contents of addressed memory location transferred to Instruction Register (instruction decoded)
  2. Instruction decoder asserts IN signal line
  3. Data in switches are transferred to A register
  5. PC incremented to 0001

  Process repeated for next instruction
- This is the Instruction Execution Cycle

CPU State Machine

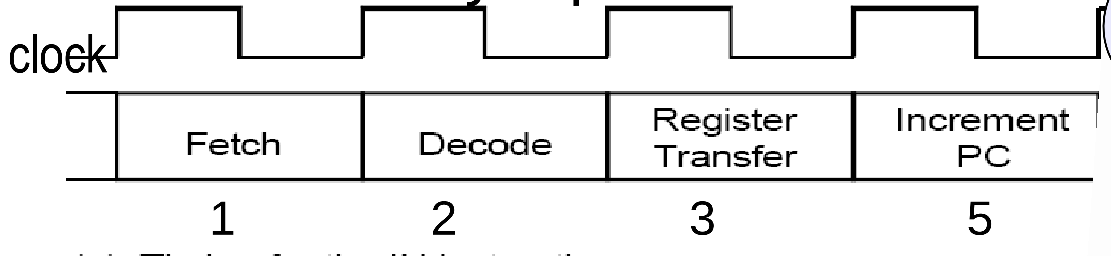
- Break down instruction into small elements - states
  - Each element-state takes an amount of time
  - Allows time for event to occur
  - Event: Propagation of signals



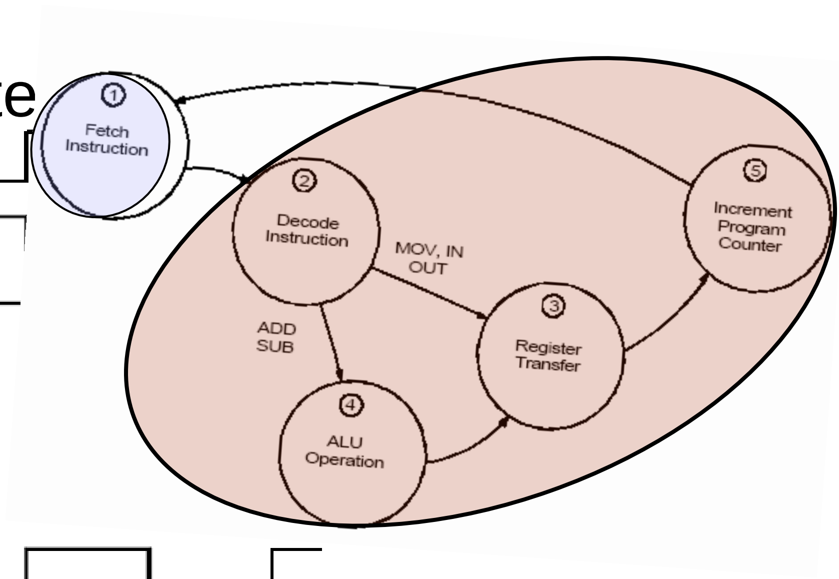
- Sequence Controller:
  - Implements state machine
  - With Instruction decoder, generates control signals at appropriate time

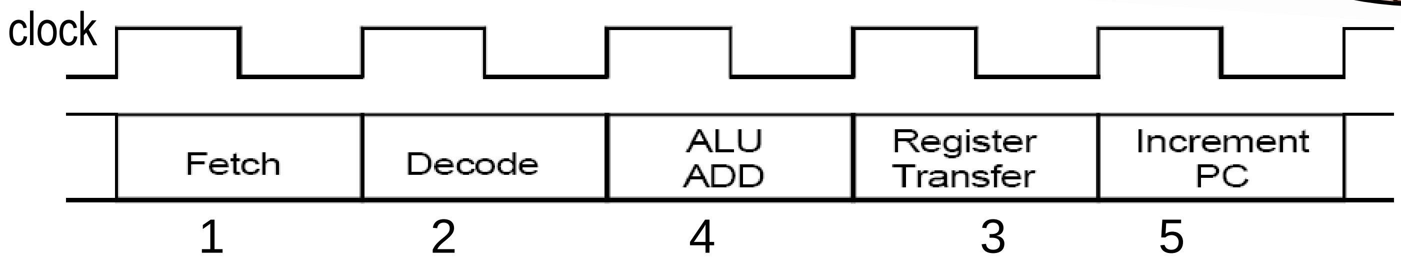
System Timing

- Sequential state machine operated by clock
  - Consider 1 MHz clock –
  - Pulse every 1 μsec for each state



(a) Timing for the IN instruction.





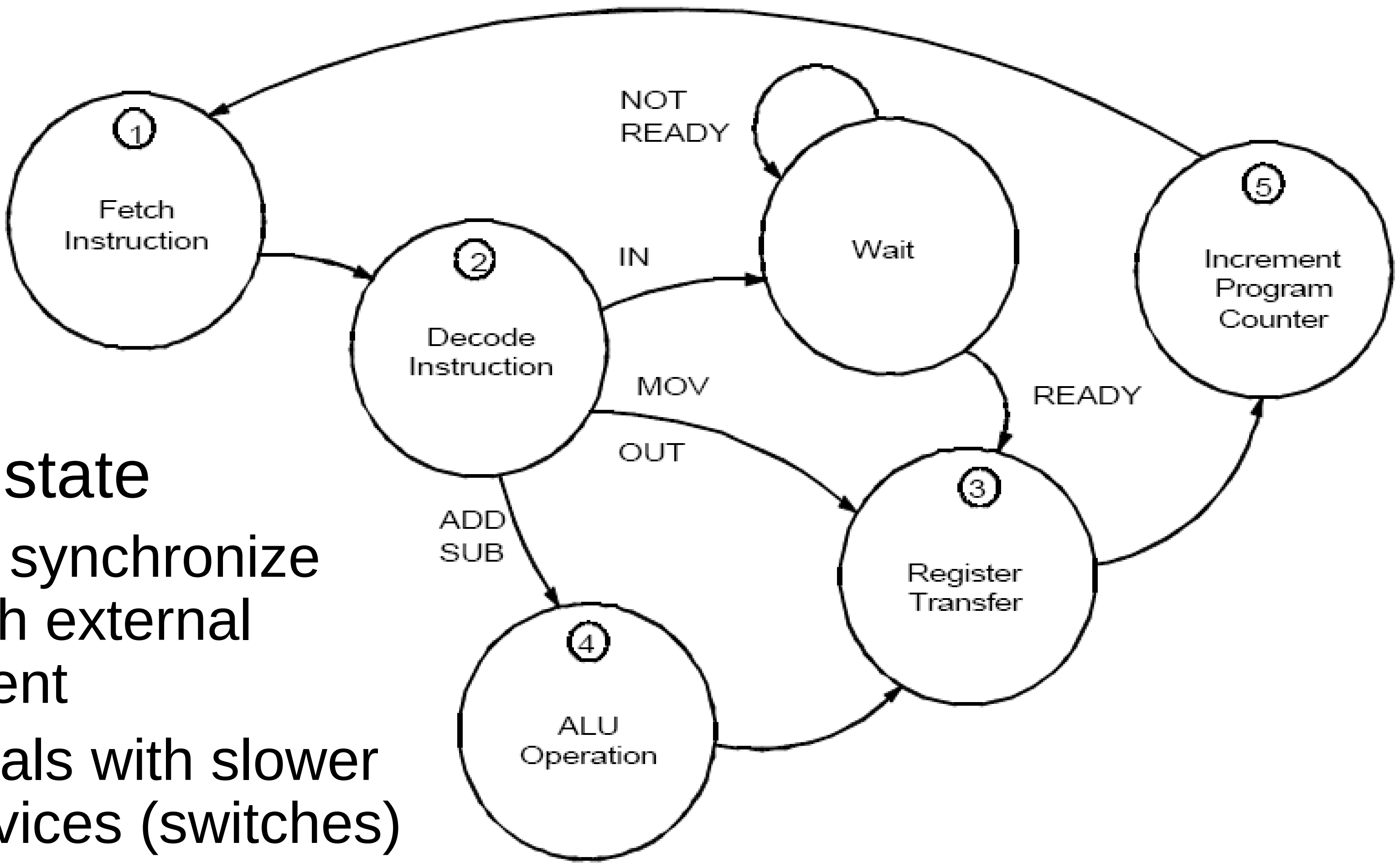
(b) Timing for the ADD instruction.

Program Execution Time

- Using 1 MHz clock, following table shows time to execute program

Instruction	Number of states
IN 1	4
MOV	4
IN 1	4
ADD	5
OUT 2	4
Total states	21 = 21 μsec

I/O Synchronization and Wait States



- Wait state
  - To synchronize with external event
  - Deals with slower devices (switches)

Improving MOV

Operation	Dest. Operand	Source Operand	
MOV	dd,	ss	(dd)←[ss]

Where dd and ss are two bit binary codes for registers

Register Codes

A = 00 B = 01 C = 10 D = 11

Example: MOV B,A ; B ← [A]

Memory Reference Instructions

- Want to be able to read data from memory and write data to memory (not only retrieve instructions)
- Addressing modes
  - Different ways to generate addresses
- Example: Immediate addressing
  - Byte *immediately* = part of op-code

MVI (move immediate) instruction

Operation	**Dest. Operand**	**Data**	
MVI	dd,	8-bit	(dd) ← (constant following op code)

2 byte instruction, where dd is two bit binary code for registers

Register Codes

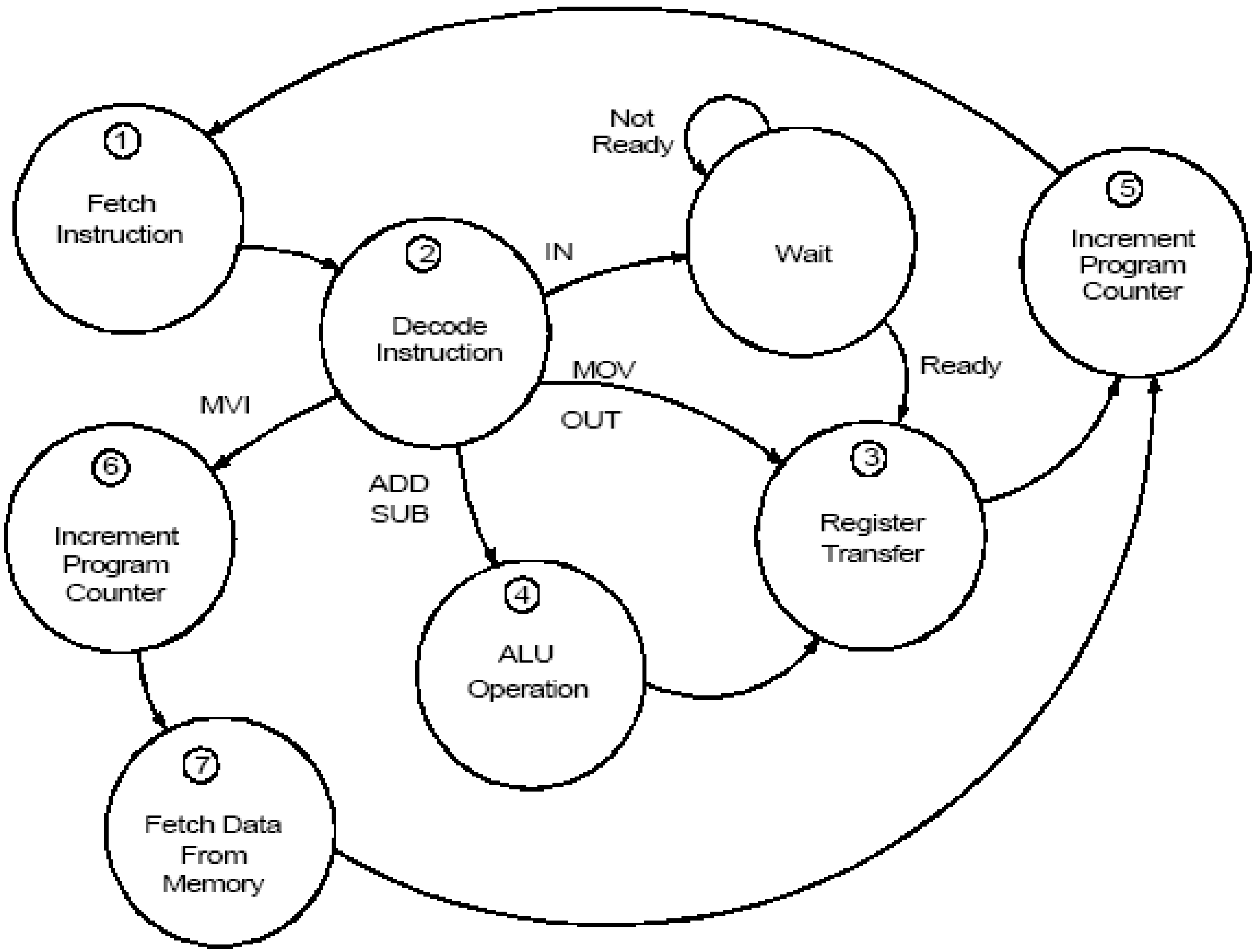
A = 00	B = 01	C = 10	D = 11

Example: MVI A,65 ; A ← 65

First byte: `110---00`

Second byte: `01000001`

State Transitions for MVI



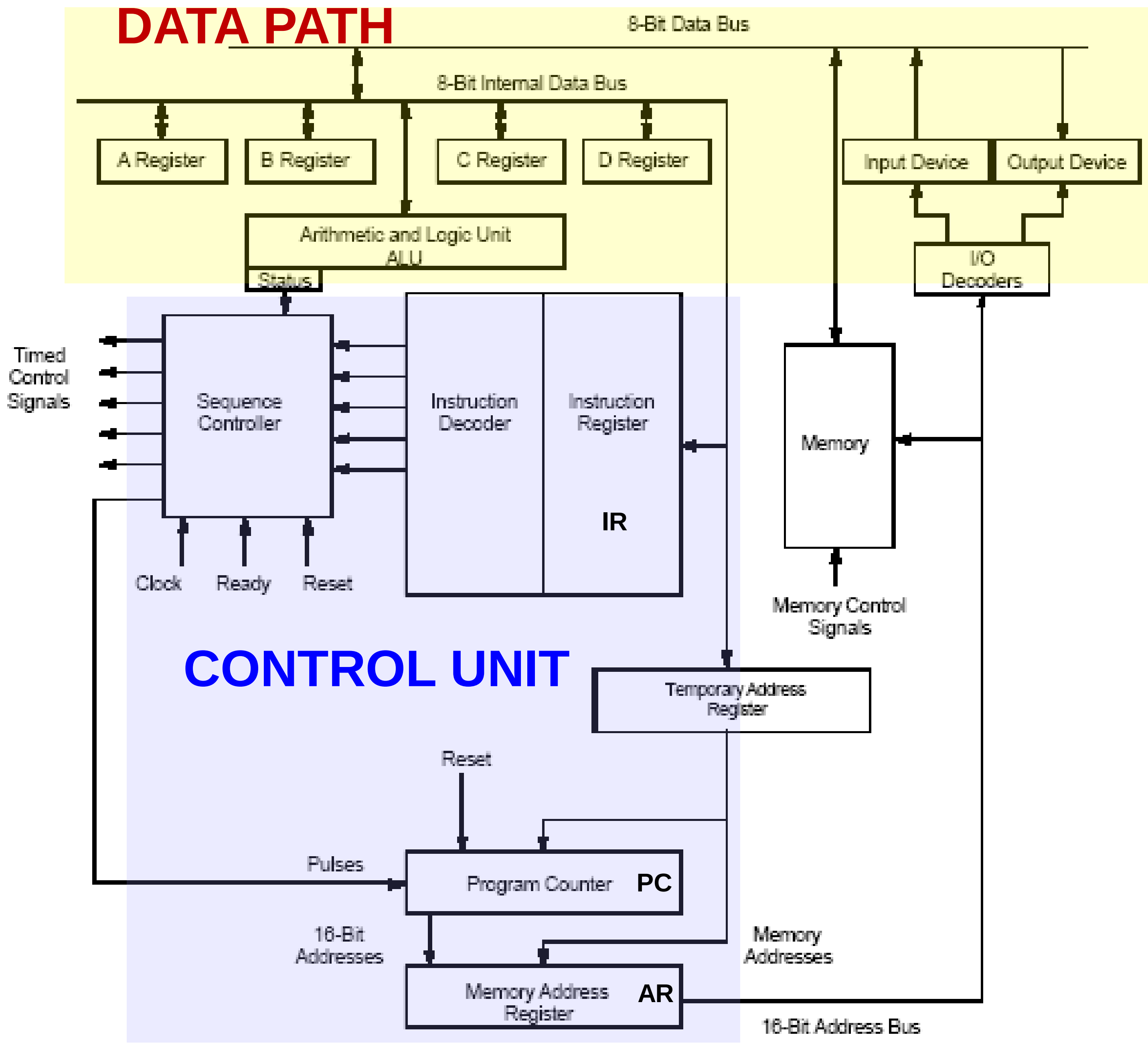
Adding Constant (65_{10}) to a Sum

Adr	Contents	Op	Opnd	Comments
0:	1010 0001	IN	1	(A) ← (Switches)
1:	0100 0100	MOV	B,A	(B) ← (A)
2:	1010 0001	IN	1	(A) ← (Switches)
3:	0010 0001	ADD	A,B	(A) ← (A) + (B)
4:	1100 0010	MVI	C,65	(C) ← (65=constant of next memory location)
5:	0100 0001			constant data: 65_{10}
6:	0010 0010	ADD	A,C	(A) ← (A) + (C)
7;	1110 0010	OUT	2	(LED) ← (A)

Control Instructions

- HALT
  - To stop processing
  - Stops pulses going to the sequence controller
- BRA (Branch always)
  - Operand is 16-bit memory address
  - Loaded into the PC
- Conditional Branching
  - Branch when condition is true, ex. Carry flag is set
  - *Carry flag* set by ALU when 8-bit add too large
  - Thus can branch when such an addition occurs
  - Carry flag, and others, found in the *Condition Code Register*

Final Design



Example of HCS12 code

- See the file Hello.lst:

```
Addr  Instr. bytes        Instr. Source
2000  CF 2000              lds #$2000
2003  CC 200B              ldd #$200B
2006  16 EE88              jsr $EE88
2009  20 F8                bra $2003
200B  48 65 6C 6C 6F 20 Data bytes for
2011  77 6F 72 6C 64 0D  'Hello world'
2017  0A 00
```

References

- Fredrick M. Cady, Microcontrollers and Microcomputers: Principles of Software and Hardware Engineering
- Fredrick M. Cady, James M. Sibigtroth; “Software and Hardware Engineering: Motorola M68HC12”

- Above are sources for most of the figures, tables and examples in the course notes