Basic Computer Organization and Design

Till Now

- We discussed registers
- We discussed the kinds of Transfer, Arithmetic, Logic and Shift Micro-operations
  - Introduced a language that connects SSI to MSI
    - (SSI: Small Scale Integration, 10-100 Gates)
    - (MSI: Medium Scale Integration, 100-1000 Gates)
    - (LSI: Large Scale Integration, 1000-10000 Gates)
  - Combined groups of instructions using multi-selection circuits
- We discussed general aspects of CPU, Memory and Bus architectures

Considering the next problem in design

- A computer organization involves combining everything we have learned to date into a single integrated unit
  - What is a computer?

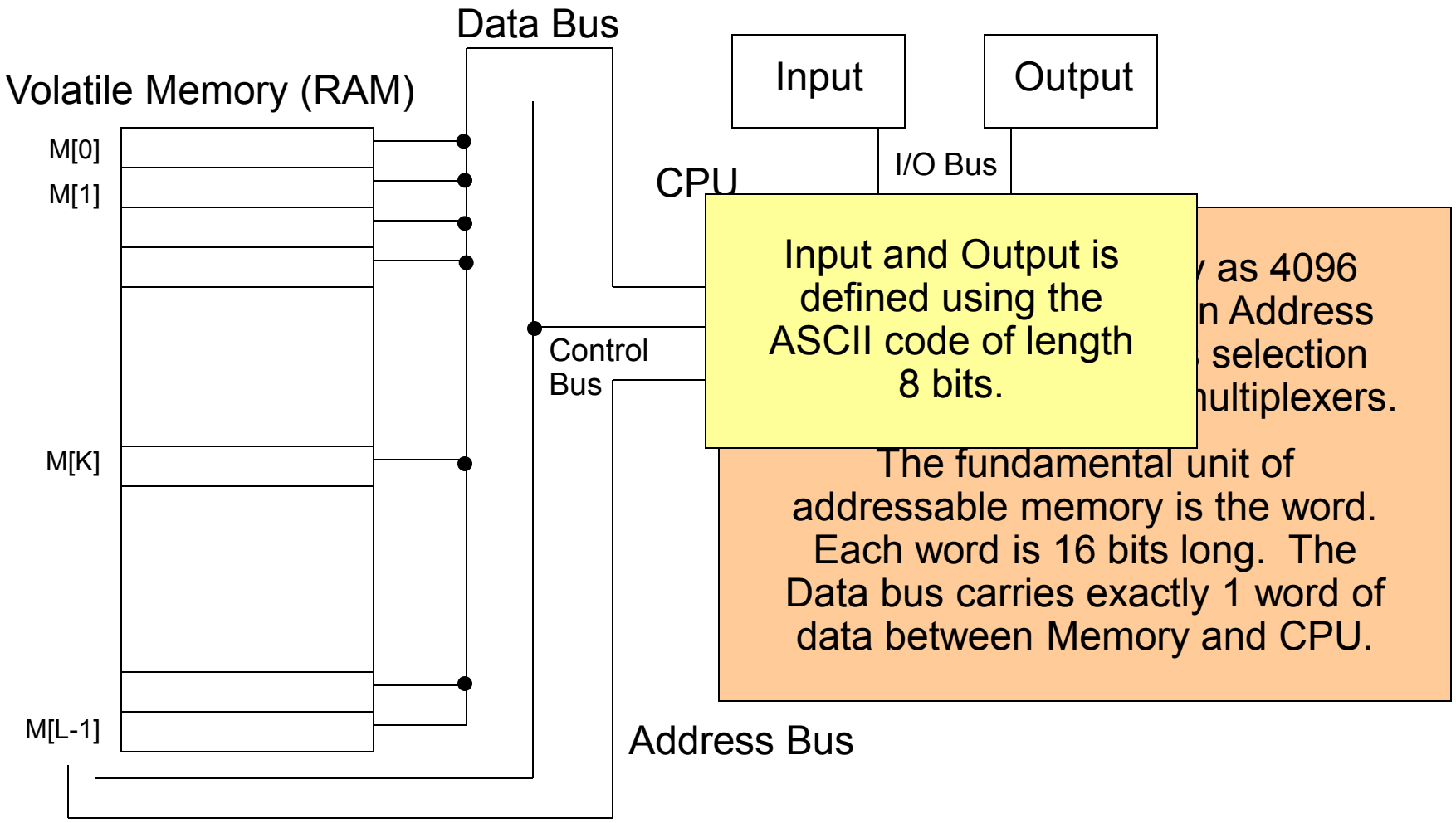
- Von Neuman refers to a computer as a "stored program digital computer"
  - What is a program?
    - What is an instruction?
    - How are instructions executed?

Goals

- We conclude our lecture series by considering computer organization
  - Combining the CPU, Memory and Bus architectures
- We introduce the concept of an instruction
  - Instruction design and architecture
  - Micro-operation sequencing (timing, control)
  - Roles of different registers
- We will follow a model of a virtual/logical computer organization adapted from M. Mano (Computer System Architecture, 3d Edition)

Computer – High Level View

- CPU, Memory and Bus architectures with interface to I/O.

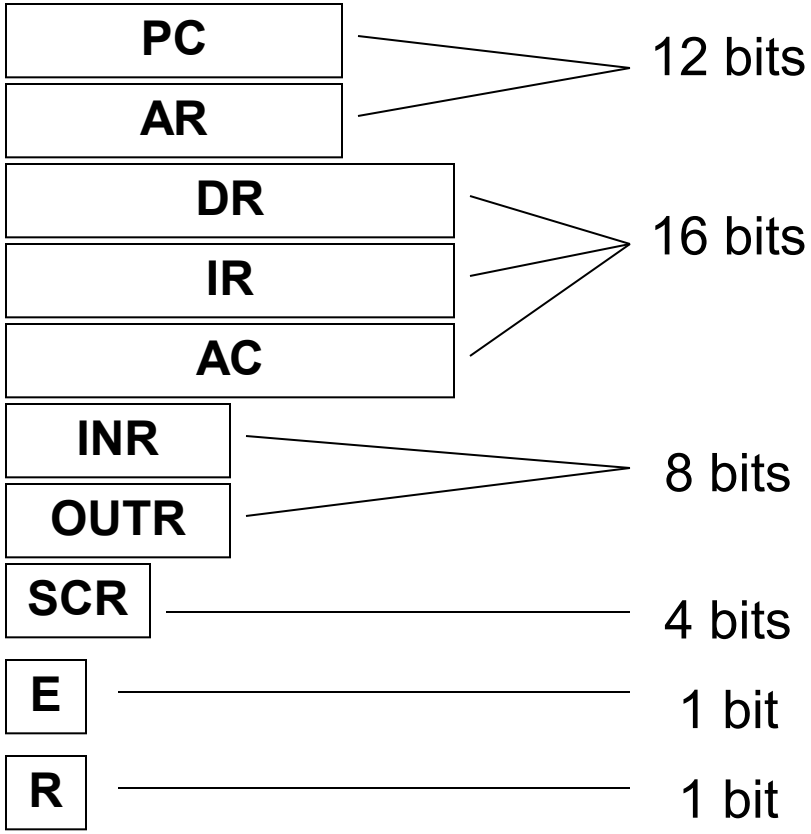


The Mano model of the CPU Registers

- CPU registers used in the model (Mano), categorized by length:
  - PC :: Program counter (address – 12 bits)
  - AR :: Address register (address – 12 bits)
  - IR :: Instruction register (data – 16 bits)
  - DR :: Data register (data – 16 bits)
  - AC :: Accumulator (data – 16 bits)
  - INR :: Input buffer register (ASCII data – 8 bits)
  - OUTR :: Output buffer register (ASCII data – 8 bits)
  - SCR :: Sequence counter register (4 bits)
  - E, R :: Single bit flip-flops (flag/utility, interrupt)

CPU Register Lengths

- The register lengths (number of flip-flops to store bits) are determined by number of memory locations (address space) and the bus width (in bits) that determines the number of bits transferred between Memory and CPU, or between I/O channels and CPU.



Stored program Concept

- A stored program is a set of instructions and data expressed in binary language, stored in non-volatile (ie. disk storage) memory

- Programs can be executed only from Memory.
  - Thus, a program must be loaded from disk to RAM in order to execute it.
  - Loaded programs are called *processes*.
  - Individual *instructions* must be transferred to the CPU where they are executed, using data that must be obtained from either CPU registers or RAM

- A process is executed by executing each individual instruction that, collectively, fulfill the intentions of the programmer.

Instruction Architecture

- The list of all instructions engineered for a computer's CPU is called the ***instruction set***.
- To distinguish each instruction, they are assigned a unique numeric code
  - Can be done in many ways
  - Does not need to be ordinal (ie. starting from 0 and running contiguously)
  - Can be partially ordinal and partially hierarchical
    - Mano's approach
- Instructions must be capable of referencing Memory and/or CPU addresses in order to cause data to be transferred and operated on.

Instruction Architecture

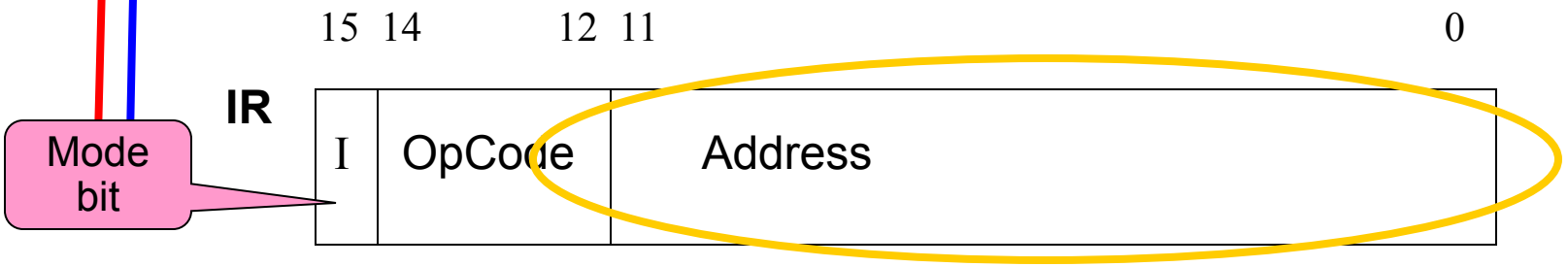
- Instructions consist of
  - Operation code data
  - Address data
  - Mode data
    - Direct addressing – data found at the specified address
    - Indirect addressing – data found at the address found at the specified address (pointer concept)

0

1

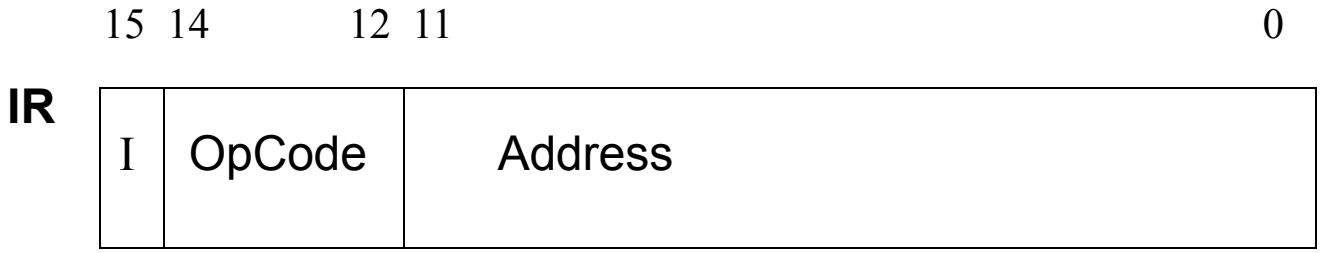
The number of memory referenced operands varies from computer to computer.

Mano simplifies the instruction architecture to permit zero or one memory address field. In the case of Indirect addressing mode there are two memory accesses, but only one reference.



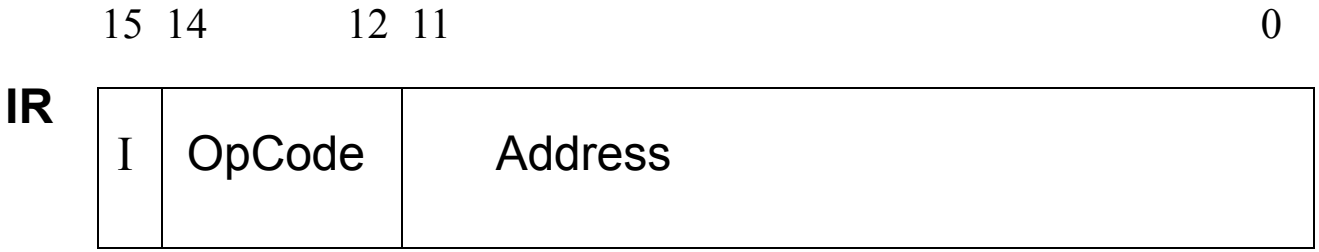
Instruction Hierarchy

- It is desirable to engineer the computer to support a reasonable number of independent instructions
  - Examples:
    - Add, subtract, transfer, shift, logic, comparison, branch, input, output
  - 4 bits can be used to support $2^4 = 16$ unique instructions
    - Leaves 12 bits to represent an address space of 4096 words
  - However, 16 instructions is not enough for most program needs



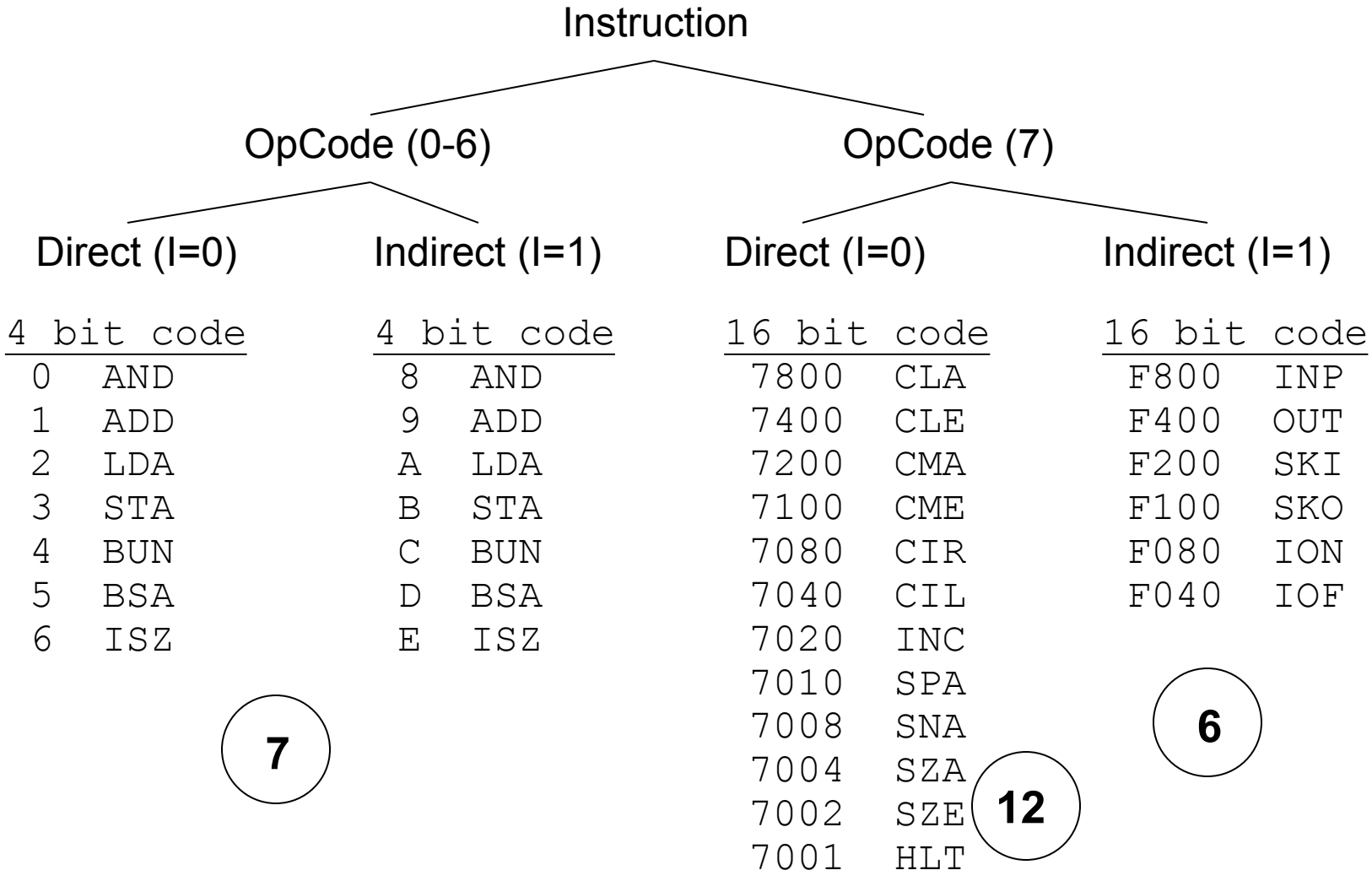
Instruction Hierarchy

- It is desirable to engineer the computer to support a reasonable number of independent instructions
  - Use a hierarchical scheme (1+3 scheme)
    - OpCode – 3 bits – supports 8 unique codes
      - 0-6 (instructions)
      - 7 (toggle)
    - Mode – 1 bit – supports 2 addressing modes
      - I = 0 Direct addressing
      - I = 1 Indirect addressing
    - Special Case :: OpCode = 7 (toggle) PLUS Mode bit (I)
      - Use 12 bits of Address field to specify **additional** instructions



Instruction Hierarchy

- Mano's instruction set consists of 25 instructions:



Instruction Set of the *Basic Computer*

Symbol	*Hex code*	*Description*
AND	0 or 8	AND M to AC
ADD	1 or 9	Add M to AC, carry to E
LDA	2 or A	Load AC from M
STA	3 or B	Store AC in M
BUN	4 or C	Branch unconditionally to m
BSA	5 or D	Save return address in m and branch to m+1
ISZ	6 or E	Increment M and skip if zero
CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right E and AC
CIL	7040	Circulate left E and AC
INC	7020	Increment AC, carry to E
SPA	7010	Skip if AC is positive
SNA	7008	Skip if AC is negative
SZA	7004	Skip if AC is zero
SZE	7002	Skip if E is zero
HLT	7001	Halt computer
INP	F800	Input information and clear flag
OUT	F400	Output information and clear flag
SKI	F200	Skip if input flag is on
SKO	F100	Skip if output flag is on
ION	F080	Turn interrupt on
IOF	F040	Turn interrupt off

Instruction Set Completeness

- Selection of instructions should span a variety of applications suitable to support programming
  - Arithmetic, logical and shift instructions
  - Instructions for moving data to and from memory and CPU registers
  - Program control instructions, instructions that check status conditions
  - Input and Output instructions

An important issue for many applications is the number of programmable (general purpose) registers.

Mano adopts a minimalist approach providing a single such register, the accumulator AC.

This simple case does allow full illustration of all computational requirements of a more powerful computer with a larger instruction set (but with much added cost to programming).

Among commercial computers one finds both complex (large number) instruction sets (CISC) and reduced (number) instruction sets (RISC).

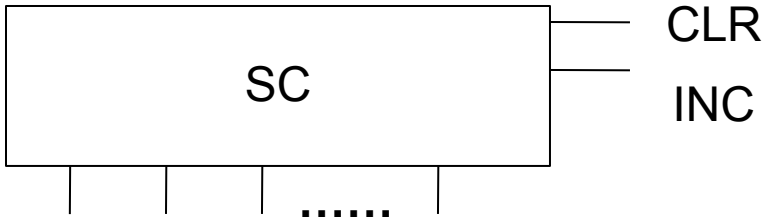
Instruction Processing & Instruction Cycle

- Instruction processing cannot occur in a single clock cycle – it is too complicated !
  - Must fetch the instruction from memory and load the instruction register, IR.
  - Must decode the Operation Code to determine whether data must be fetched and whether we fetch data directly, or indirectly
  - Must fetch required data, if any, from memory into CPU
    - OR, must store data from CPU into memory
    - OR, must obtain data from Input
    - OR, must place data for Output
  - Must process the data according to instruction logic
- Thus, the ***instruction cycle*** consists of a controlled sequence of microoperations

Timing and Control

- In order to control the steps of the instruction cycle, it is necessary to introduce a counter, whose output is used as input to the control logic

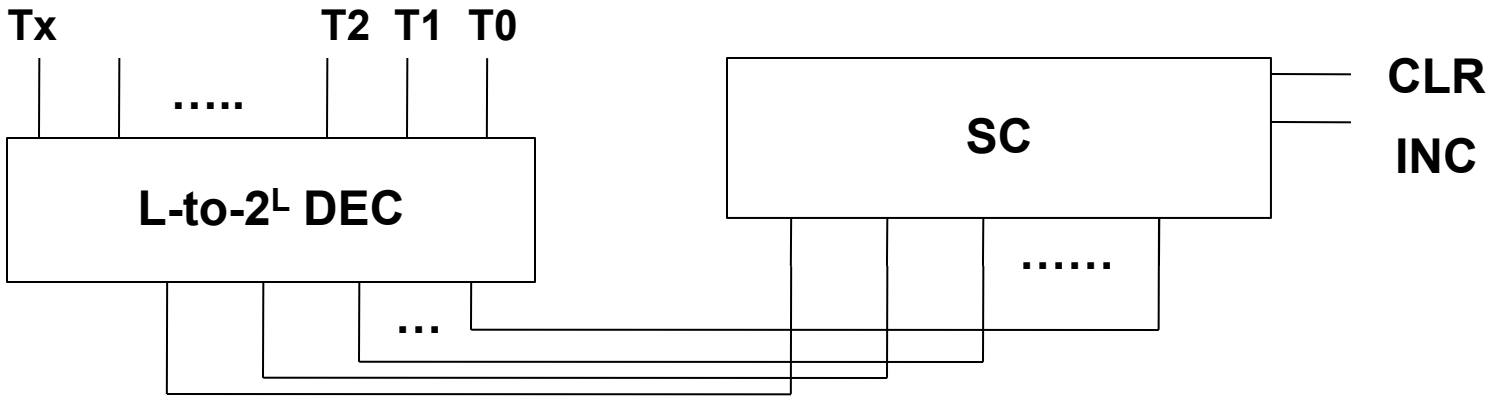
  – We will utilize a Sequence Counter Register (SC, or SCR). This is a register that holds a count value, can be reset to zero and can be incremented (or decremented) by one



  – Each instruction will require a specified number of time steps to complete a sequence of micro-operations. Each step of the sequence is marked by a count value in SC.

Timing and Control

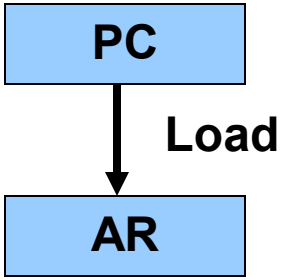
- The SC outputs a string of bits whose value is in the range from 0 to 2^L-1
  - Eg. for L=3, from 0 to 7
- We need a way of converting the bit string value to single bit valued outputs labelled T0, T1, T2, T3, and so on, up to Tx (where x = 2^L-1)
- A decoder serves our purpose, recalling that the output from the DEC is a 1 only on one line (the rest are 0`s)



Timing and Control

- Step 1: Where do we fetch the next instruction from at **T0** (start time)?
  - The **Program Counter register (PC)** is assumed to hold the address of the next instruction to be executed.
    - It is loaded initially during the machine bootstrapping process.
  - The PC is transferred to the Address Register (AR) which is connected to the Address Bus Multiplexer.
    - The AR provides the selection input to the address bus multiplexer to enable RAM-CPU data transfers

```
T0 :  AR = PC
```



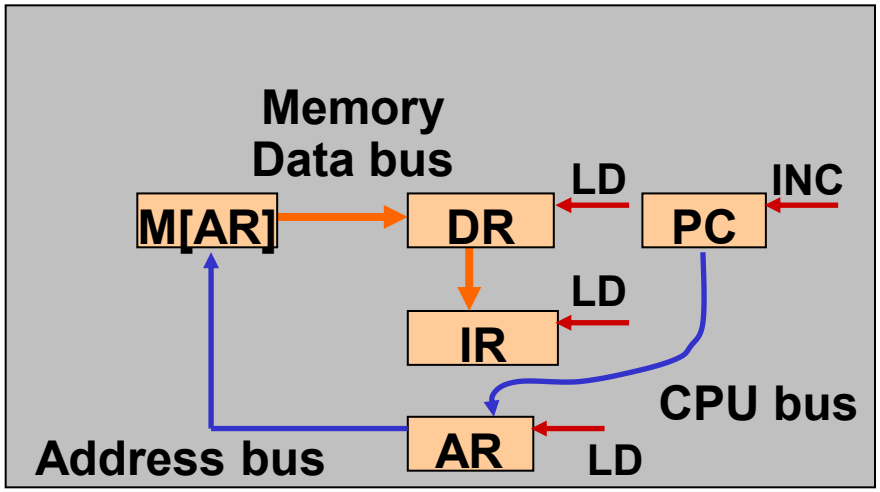
Timing and Control

- Step 2: How do we obtain the instruction from memory at **T1**?
  - The data stored at the selected address is transferred onto the Data Bus and then to the **Data Register (DR)** in the CPU.
  - The instruction data is transferred immediately to the **Instruction Register (IR)**

Note how the PC is updated in parallel. This assumes that the next instruction (yet to be executed) in logical sequence is actually located at the next memory location. This restriction to non-sequential instructions is removed by including instructions that modify the PC explicitly.

```
T1 :  DR = M[AR] , IR = DR , PC = PC + 1
```

NOTE: Mano does not include this step explicitly. However, in most systems the initial transfer from memory is always to a CPU buffer, such as DR. In effect, this is simply a *pipeline* element of a transfer circuit that adds an intermediate parallel load through DR to IR. This can be modified to act as a Master-Slave element which is necessary to build a Direct Memory Access (DMA) control circuit.



Timing and Control

- Step 3: How do we decode the instruction *intention (ie. meaning, operational definition)* at **T2**?
  - The OpCode, Mode and Address field bits all serve as inputs to the Control Logic Gates that select the specific instruction semantics
    - Direct addressing
    - Indirect addressing
    - CPU register addressing
    - Input
    - Output

NOTE: All instructions share the steps T0, T1 and T2. However, depending on what each specific instruction requires, the control logic for higher time steps is more complicated, requiring more inputs.

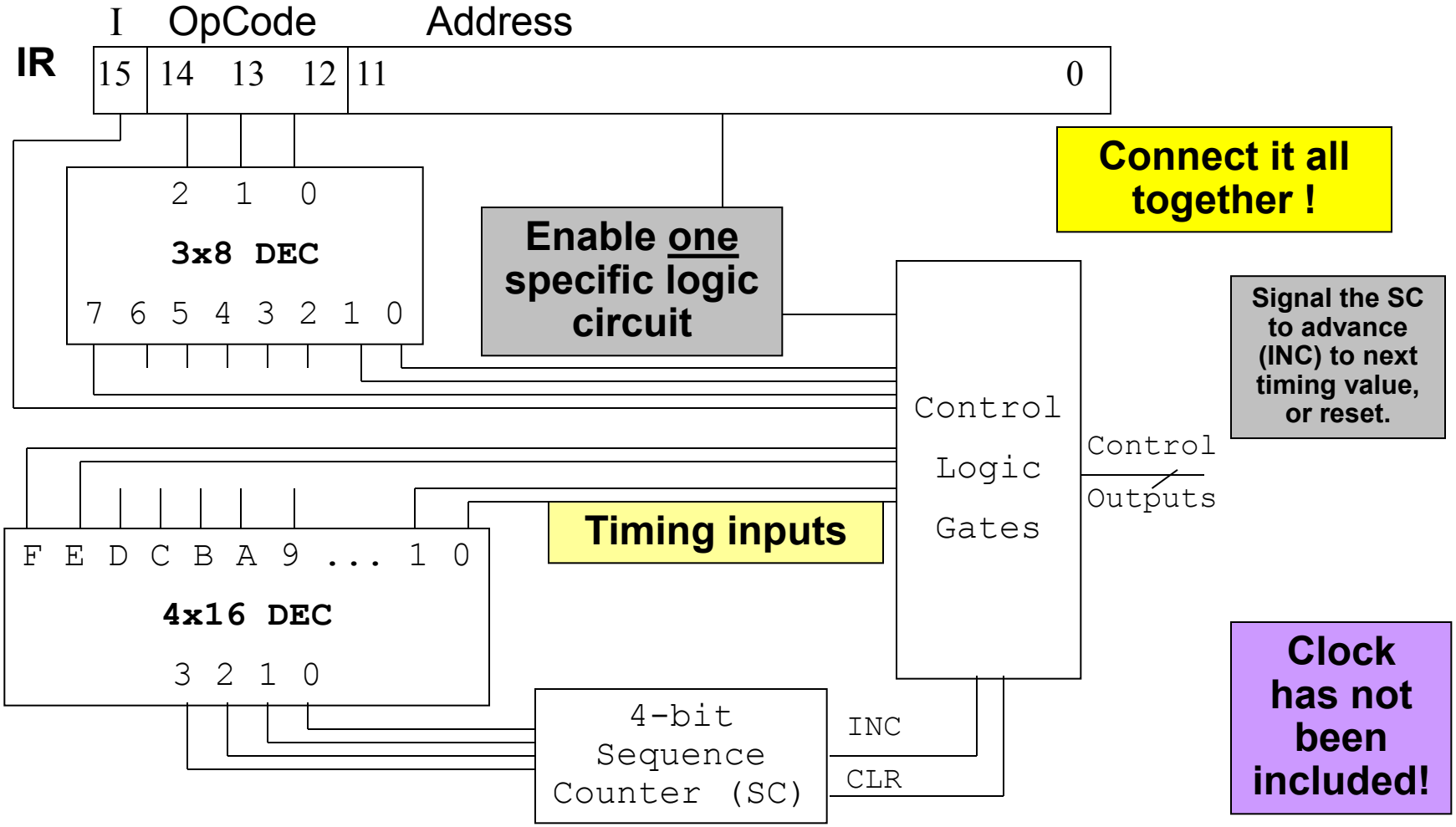
Note the reference to a DECoder unit to explicitly access the IR bits indicated

```
T2 : {D0,...,D7} = DEC( IR(12-14) ),
     AR = IR(0-11),
     I = IR(15)
```

The AR may be directly connected to the IR address bits indicated

Timing and Control

- The ***instruction cycle*** consists of a controlled sequence of micro-operations using control logic gates and a sequence counter.



An Architectural Basis for Programming

- **Programs are written by humans and typically use high-level languages to express logic.**
  - Programmers learn to use languages based on expressiveness and ease of use
  - Language designers are not limited by imagination in developing new operations and techniques
  - But, too much complexity presents real difficulties for practical, widespread usage.

- **Computer Instruction Sets are limited by the complexities of**
  - Working with materials on a micro- and nano-scale of fabrication
  - Understanding of fundamental physics applied to signaling and electromagnetic properties of matter
  - Optimal design of complicated circuits with well-posed input/output logics
  - Power efficiency issues, temperature control
  - Performance and timing considerations
  - Many other practical and theoretical objectives

An Architectural Basis for Programming

- **Thus, Instruction Sets are minimal reflections (subduced mappings) of high-level operation and function sets**
  - Limited number of hardware operations
    - Assignment = Transfer
    - Integer arithmetic (may include optimized floating point)
    - Logic, Shift
    - Non-sequential control (branching for decision and repetition)
    - Input and Output
    - Communications
    - Specialized operations (string, array)
    - Specialized addressing modes
    - Enhanced data and register access modes

Instruction Categories

- **Instructions may be placed in several different kinds of categories**
  - Conceptual application, or meaning
  - Performance based on location of operation
    - Studies of application programs have found that most programs perform best when optimized for use of the CPU (called CPU intensive jobs)
    - However, many programs must interact with peripheral devices that operate on relatively slow electromechanical hardware (called I/O intensive jobs)
    - Typically, programs are hybrids that must access I/O, memory and CPU in various combinations and for different intervals of intensive use of a particular resource.

- **We adopt a performance based on location categorization of instructions**
  - Register reference
  - Memory reference
  - I/O reference

  - Within each category above we define conceptual subcategories

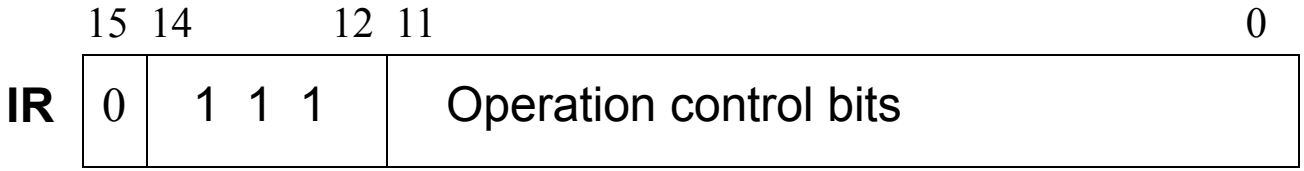
Register Reference Instructions

- Referencing CPU registers for data processing is the fastest data access possible

  – We consider two registers

    - The **Accumulator (AC)**
      – Used for arithmetic, logic and shift
      – Also used for input and output

    - The **E-bit (E)** utility flip-flop
      – Used in shift operations
      – May be used in arithmetic operations



Register Reference Instructions

- The complete set of all instructions defined by Mano
  - *Mnemonic* means a string that *suggests* the meaning.

Hex	Mnemonic	Binary coding	Meaning
7800	CLA	0111100000000000	CLear Accumulator
7400	CLE	0111010000000000	CLear E
7200	CMA	0111001000000000	CoMplement Accumulator
7100	CME	0111000100000000	CoMplement E
7080	CIR	0111000010000000	CIrcular shift Right
7040	CIL	0111000001000000	CIrcular shift Left
7020	INC	0111000000100000	INCrement AC
7010	SPA	0111000000010000	Skip if Positive Accumulator
7008	SNA	0111000000001000	Skip if Negative Accumulator
7004	SZA	0111000000000100	Skip if Zero Accumulator
7002	SZE	0111000000000010	Skip if Zero E
7001	HLT	0111000000000001	HaLT the computer

Register Reference Instructions

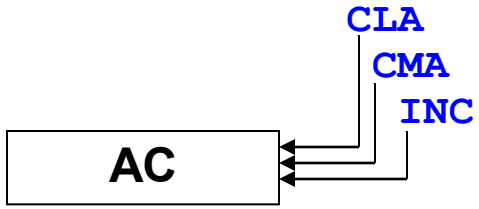
- Within this instruction subset, one may find it convenient to group the instructions by which register is affected

Hex	Mnemonic	
7800	CLA	AC affected
7200	CMA	
7080	CIR	
7040	CIL	
7020	INC	
7010	SPA	
7008	SNA	
7004	SZA	
7400	CLE	E affected
7100	CME	
7002	SZE	
7001	HLT	Control affected

Register Reference Instructions

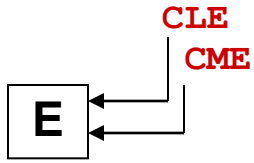
- Clear, Complement or Increment : AC register.

Hex	Mnemonic	RTL
7800	CLA	AC = 0
7200	CMA	AC = ~ AC
7020	INC	AC = AC + 1



- Clear, Complement : E register

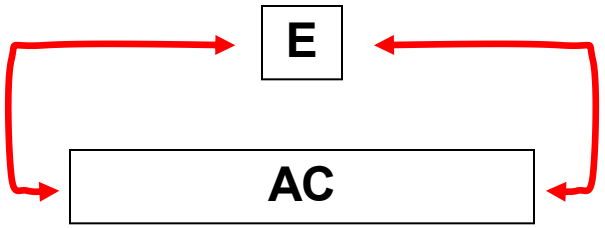
Hex	Mnemonic	RTL
7400	CLE	E = 0
7100	CME	E = ~ E



Register Reference Instructions

- Shift AC register.
  - This is a **circular** shift that is performed using the **E** register
  - Control over timing ensures all operations operate in parallel
    - Eg. Use master-slave flip-flops in registers

Hex	Mnemonic	RTL
7080	CIR	AC(0-14) = AC(1-15), AC(15) = E, E = AC(0)
7040	CIL	AC(1-15) = AC(0-14), AC(0) = E, E = AC(15)



Register Reference Instructions

- Skip on <condition> : AC register.
  - Tests sign/value status of 2's complement integer in AC
  - If status matches query, advance PC by one instruction word

Hex	Mnemonic	RTL
7010	SPA	(AC > 0) : PC = PC + 1
7008	SNA	(AC < 0) : PC = PC + 1
7004	SZA	(AC = 0) : PC = PC + 1

- Skip on Zero condition : E register
  - Test status of E bit. If not zero, proceed to the next instruction.
  - If zero, advance PC by one instruction word (recall that it has already been incremented by one, so this causes skipping the next instruction in contiguous sequence.

Hex	Mnemonic	RTL
7002	SZE	~ E : PC = PC + 1

Register Reference Instructions

- Halting the computer
  - Disable all circuits (over-ride all specific Enable controls with a general Disable).

Hex	Mnemonic	RTL
7001	HLT	Disable all circuits

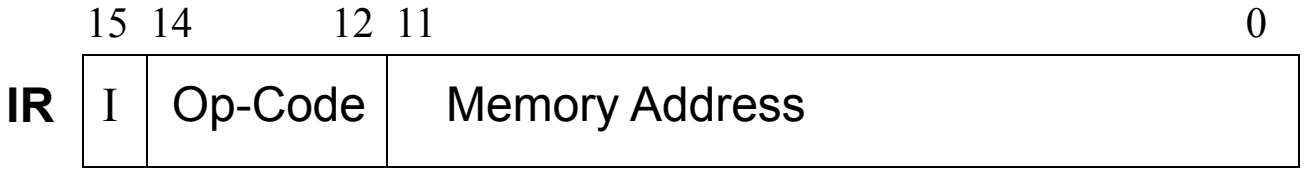
- Bootstrapping the computer
  - "Turning on" the computer simply refers to supplying electricity to the circuits and to the clock
  - Since no instruction may be fetched to IR for execution unless a RAM address is specified for the instruction, it is necessary to explicitly load the PC with an initial instruction address.
  - The textbook does not discuss this matter very much. To state it briefly, a special memory called a ROM (a non-volatile memory device) contains a sequence of instructions that loads an operating system program, from disk into memory, and with the PC set to an initial instruction address.

Memory Reference Instructions

- Once an instruction has been loaded to IR, it may require further access to memory to perform its intended function
  - Direct Access – access the storage at the specified IR address
  - Indirect Access – first, transfer to DR the data stored at the specified IR address, then (after transferring DR to AR), access this fetched address to perform the required function.

Operation Codes

Direct	Indirect	
I=0	I=1	Mnemonic
0	8	AND
1	9	ADD
2	A	LDA
3	B	STA
4	C	BUN
5	D	BSA
6	E	ISZ



Memory Reference Instructions

- Direct access mode (I=0) operation semantics

The actual number of timing steps (Tk) may vary from 1 to 2 (or even 3) steps, depending on how the control timing circuits are designed.

Operation Codes

Direct

I=0	Mnemonic	RTL
0	AND	DR = M[AR] ; AC = AC ^ DR
1	ADD	DR = M[AR] ; AC = AC + DR
2	LDA	DR = M[AR] ; AC = DR
3	STA	DR = AC ; M[AR] = DR
4	BUN	AR = IR(0-11) ; PC = AR
5	BSA	AR = IR(0-11) ; M[AR] = PC , PC = AR + 1
6	ISZ	AR = IR(0-11) ; DR = M[AR] ; AC = DR ; AC = AC + 1 ; DR = AC ; M[AR] = DR , (AC=0) : PC = PC + 1

Memory Reference Instructions

- Indirect access mode (I=1) operation semantics using AND and ADD
  - First, fetch the pointer (address) data from memory to obtain the indirect reference
  - Second, fetch the needed data from the just-fetched address

The actual number of timing steps (Tk) may vary from 2 to 3 steps, depending on how the control timing circuits are designed.

Operation Codes

Indirect

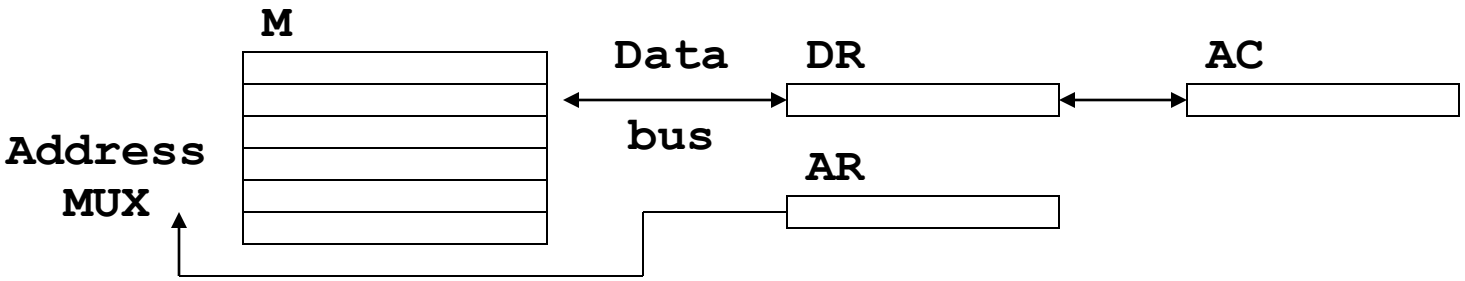
I=1	Mnemonic	RTL
8	AND	AR = IR(0-11) ; DR = M[AR] AR = DR(0-11) ; DR = M[AR] AC = AC ^ DR
9	ADD	AR = IR(0-11) ; DR = M[AR] AR = DR(0-11) ; DR = M[AR] AC = AC + DR

Memory Reference Instructions

- Transfer instructions
  - Copy data to or from memory and CPU
    - Loading (fetching) refers to memory-to-CPU transfers
    - Storing refers to CPU-to-memory transfers

Operation Codes

Direct I=0	Indirect I=1	Mnemonic	RTL
2	A	LDA	DR = M[AR] ; AC = DR
3	B	STA	DR = AC ; M[AR] = DR



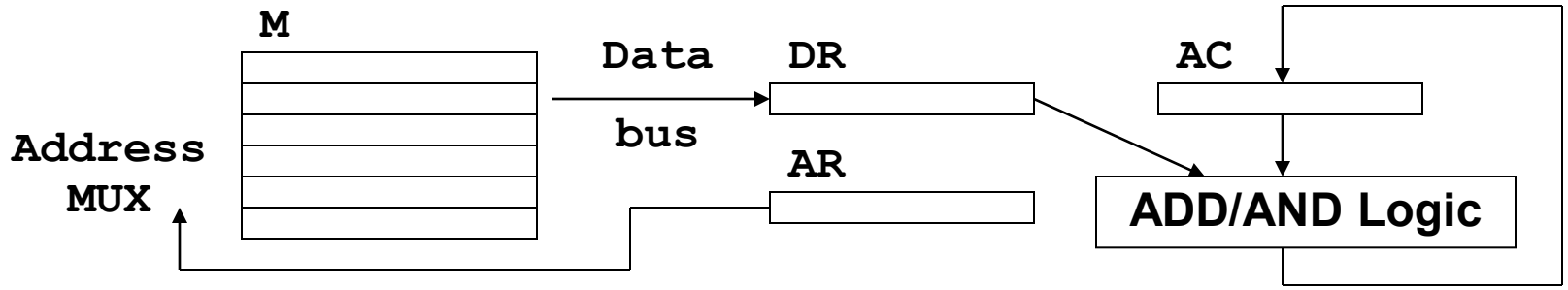
Memory Reference Instructions

- ADDition, Logical AND instructions
  - Logical AND is done on all bits in parallel
  - Numeric (2's complement) addition may be done in serial, or in parallel using Look-Ahead circuits

Operation Codes

Direct Indirect

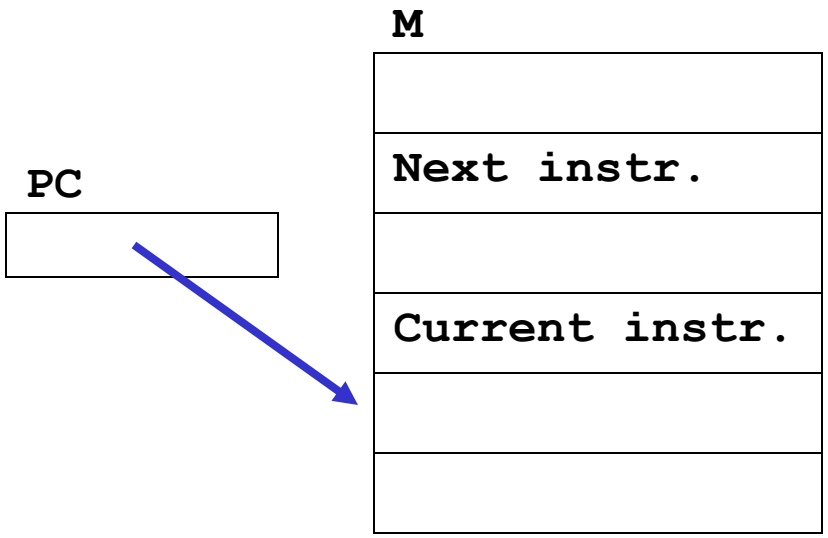
I=0	I=1	Mnemonic	RTL
0	8	AND	DR = M[AR] ; AC = AC ^ DR
1	9	ADD	DR = M[AR] ; AC = AC + DR



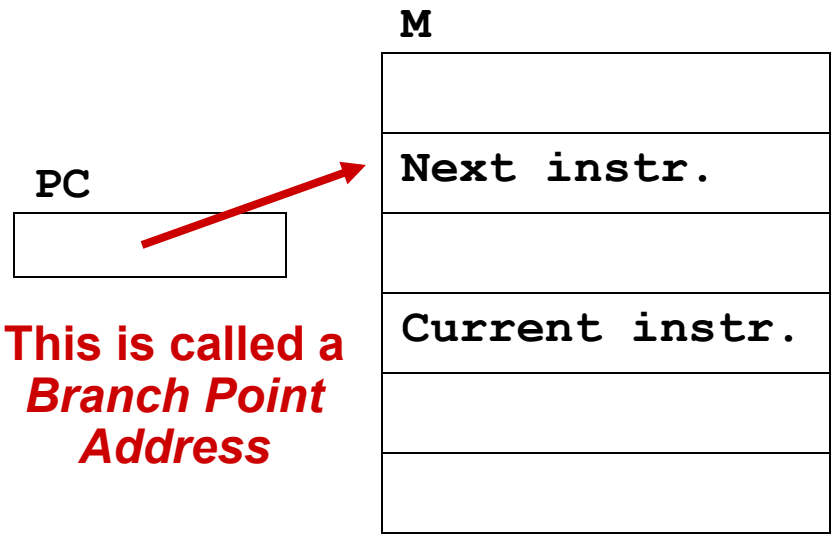
Memory Reference Instructions

- Branching
  - Instructions that explicitly modify the PC during execution
  - This implies that the *next* instruction to be executed is **not** located in the next contiguous word of storage.
  - Used to program IF-THEN-ELSE (Decision control) and WHILE-DO (Repetition control) constructs in high-level languages.

(a) BEFORE COMPLETION OF EXECUTION (after DECoding)



(b) AFTER COMPLETION OF EXECUTION



This is called a Branch Point Address

Memory Reference Instructions

- Timing and control logic must be carefully worked out for all steps in the instruction.
  - For example, using the AND instruction:

T0: AR = PC, SC = SC + 1

T1: IR = M[AR], PC = PC +1, SC = SC +1

T2: {D0..D7} = DEC(IR(12..14)), AR = IR(0..11), I = IR(15), SC = SC+1

D0.T3: DR = M[AR], SC = SC+1

I'.D0.T4: AC = AC^DR, SC = 0

I.D0.T4: AR = DR(0..11), SC = SC+1

I.D0.T5: DR = M[AR], SC = SC+1

I.D0.T6: AC = AC^DR, SC = 0

  - Note how the Direct and Indirect forms of the instruction are differentiated using the I (or I') bit value for control differentiation.

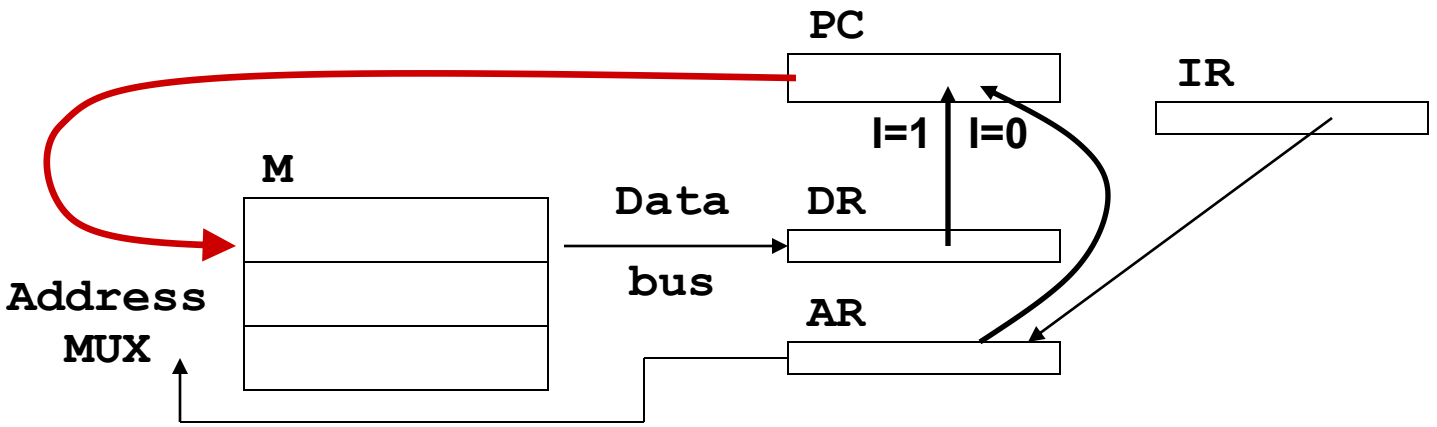
Memory Reference Instructions

- Branch UNconditional instruction
  - I = 0 :: Replace PC by address in IR(0-11)
  - I = 1 :: Replace PC by the address found at the address in IR(0-11)
    - That is :: **PC = M[M[IR(0-11)]]**

Operation Codes

Direct Indirect

I=0	I=1	Mnemonic	RTL
4	C	BUN	AR = IR(0-11) PC = AR

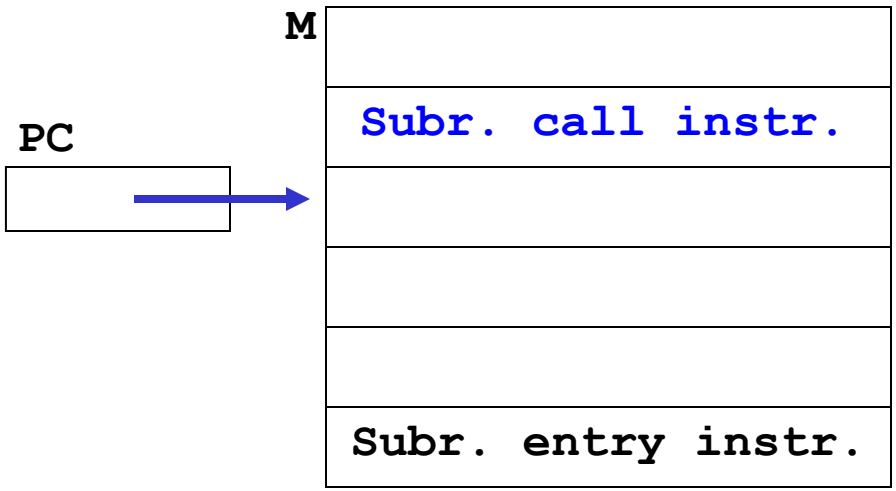


Memory Reference Instructions

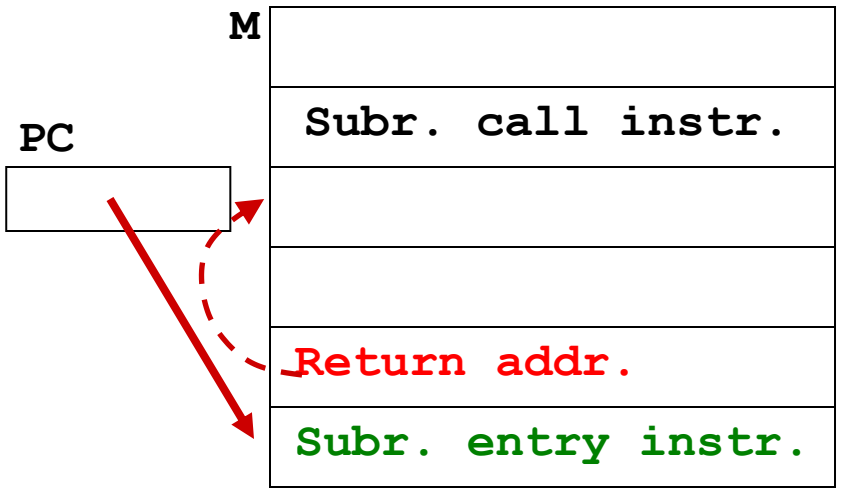
- **B**ranch and **SA**ve instruction
  - Used for programming subroutine calls

Operation Codes			
Direct	Indirect		
I=0	I=1	Mnemonic	RTL
5	D	BSA	AR = IR(0-11) M[AR] = PC , PC = AR + 1

(a) BEFORE SUBROUTINE CALL



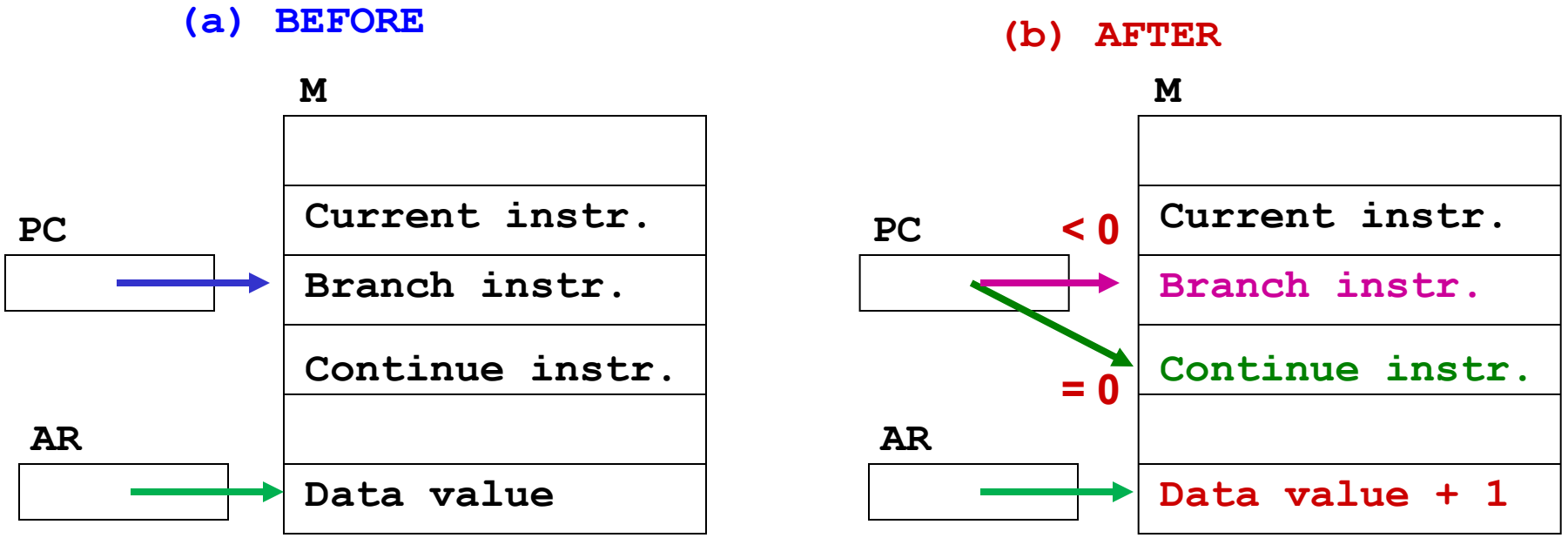
(b) AFTER SUBROUTINE CALL



Memory Reference Instructions

- Increment and Skip if Zero instruction
  - Used to implement a *counter based <do-while>* construct

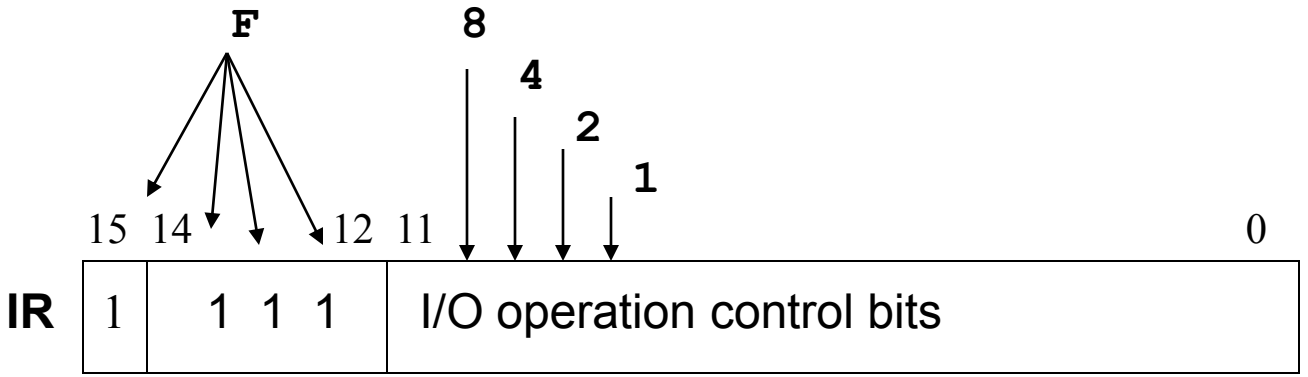
I=0	I=1	Mnemonic	RTL
6	E	ISZ	AR = IR(0-11) , DR = M[AR] AC = DR , AC = AC + 1 , DR = AC M[AR] = DR , (AC=0) : PC = PC + 1



Input/Output Instructions

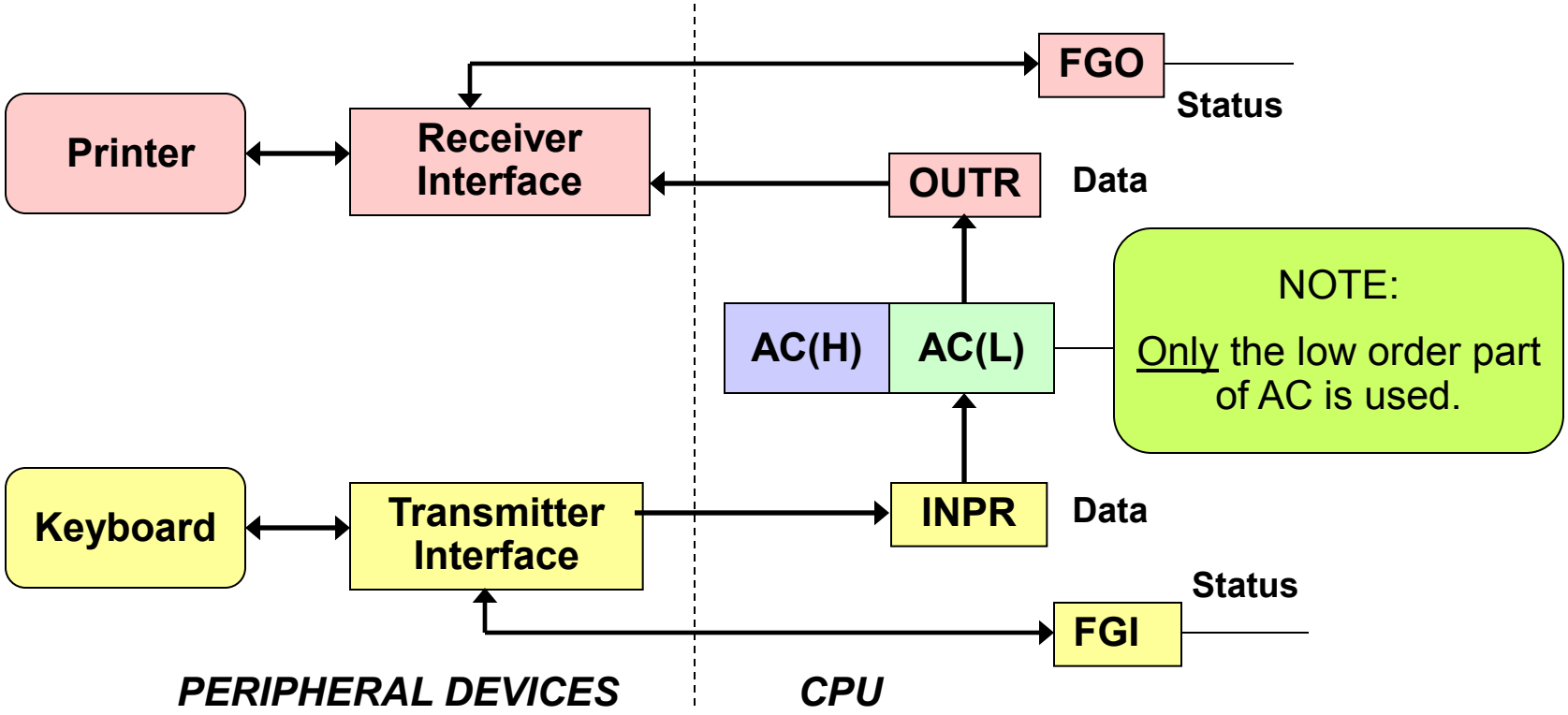
- Used for communicating data between CPU and I/O peripheral devices
- Also, need instructions to support programmed *polling*.
  - Polling refers to waiting for a condition to be true before proceeding

16 bit OpCode	Mnemonic	Meaning
F800	INP	Input ASCII char
F400	OUT	Output ASCII char
F200	SKI	Skip if input flag (FGI=1)
F100	SKO	Skip if output flag (FGO=1)



Input/Output Instructions

- Each peripheral device has a communications and control interface that interacts with the computer's interface logic circuits
  - Input
    - Need a data buffer (INPR) and a flag (FGI) indicating buffer empty/full
  - Output
    - Need a data buffer (OUTR) and a flag (FGO) indicating buffer empty/full

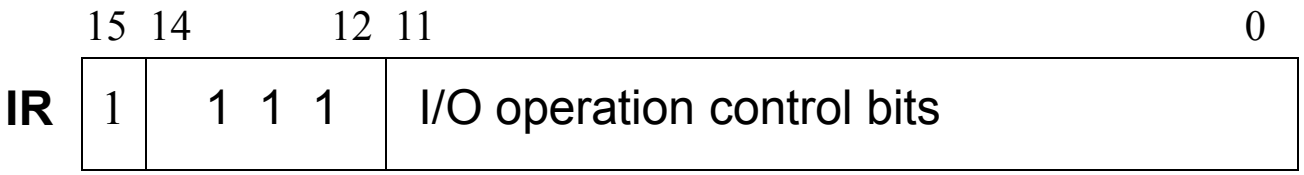


Interrupts

- Input and Output interactions with electromechanical peripheral devices require huge processing times compared with CPU processing times
  - I/O (milliseconds) versus CPU (nano/micro-seconds)
- Interrupts permit other CPU instructions to execute while waiting for I/O to complete
  - Need an additional 1-bit **IEN** flip-flop to store the interrupt status (0/1)
  - IEN – Interrupt Enable

16 bit OpCode	Mnemonic	Meaning
F080	ION	Interrupt Enabled (IEN ← 1)
F040	IOF	Interrupt Disabled (IEN ← 0)

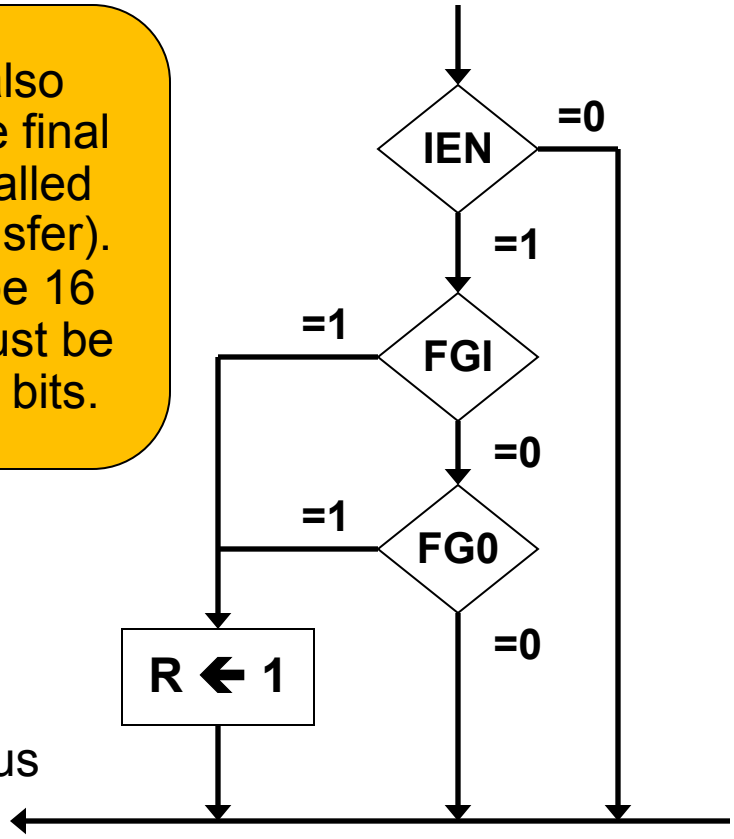
IEN



External Interrupts Handling Chart

- In this approach, interrupts are used only with I/O handling
  - In addition to a flip-flop to store the Interrupt Enable (IEN) state, one more flip-flop (R) is needed to store the I/O Status (Ready/Not ready).
  - In general, interrupts may be used with arbitrary instructions for exception trapping and handling

We will also require one final register, called **`TR`** (for transfer). This can be 16 bits, but must be at least 12 bits.



1-bit registers

IEN

FGI

FGO

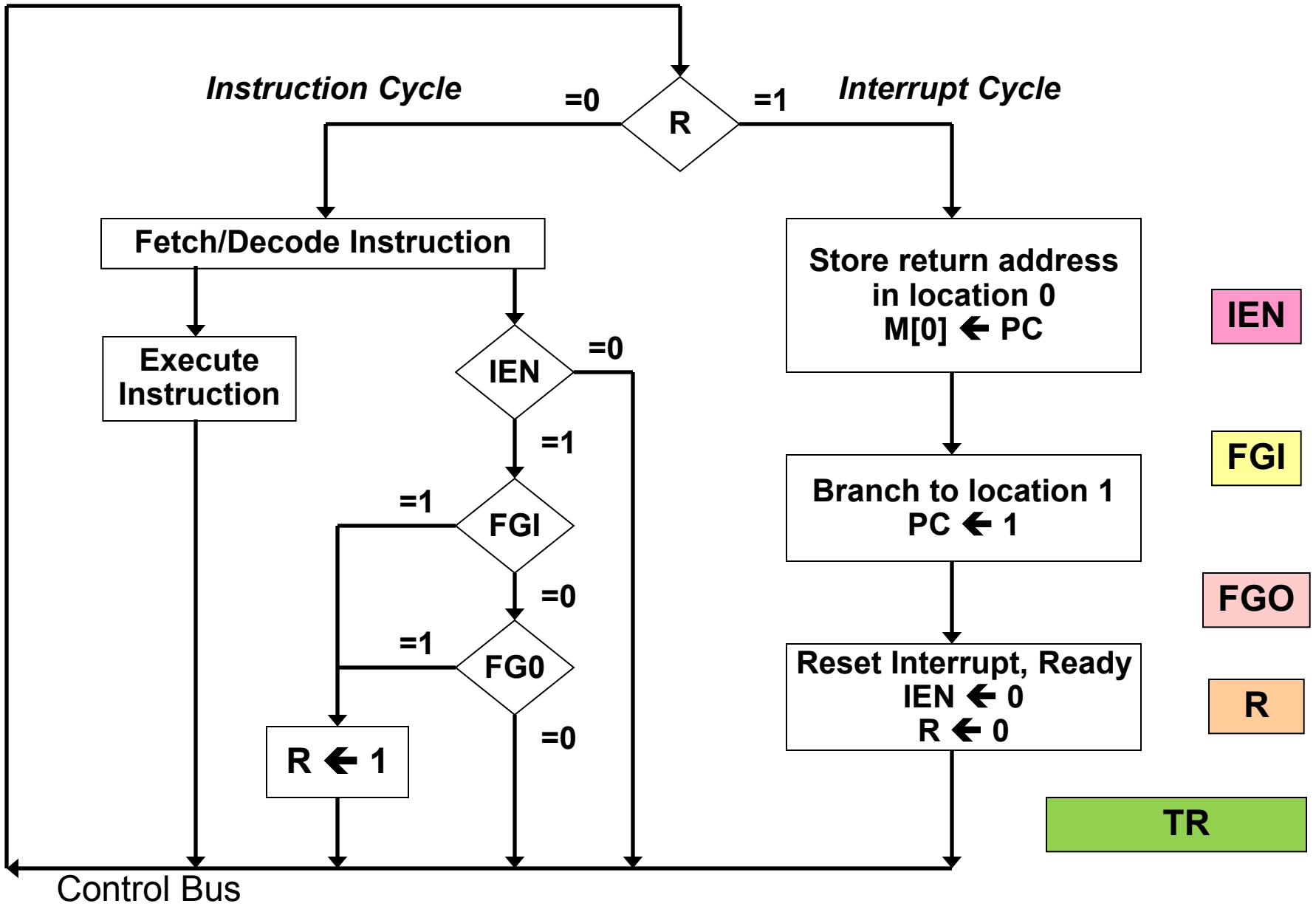
R

All of these flip flops are assumed to be reset to 0 when bootstrapping the computer.

FGI - Input Buffer Flag

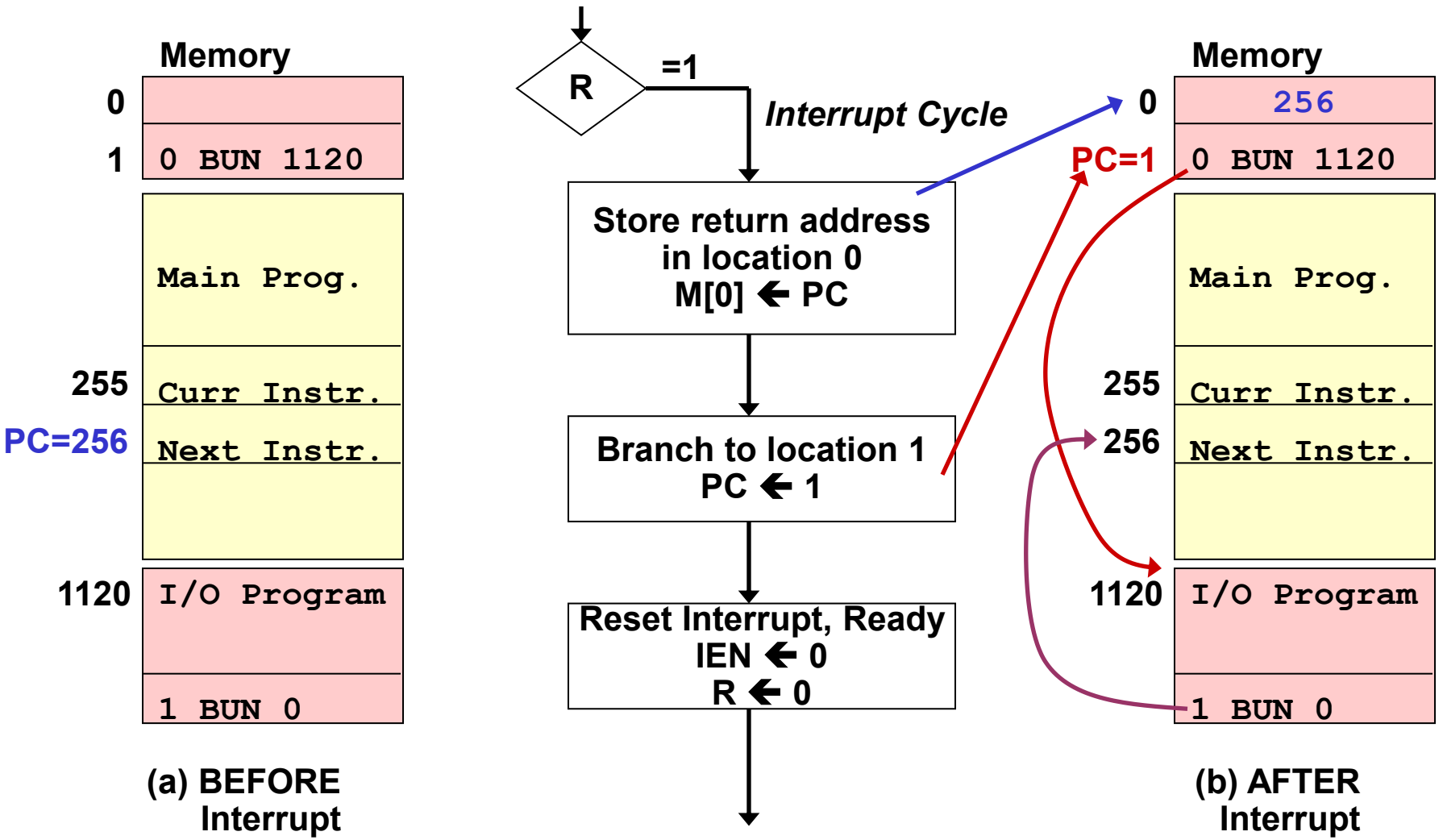
FGO - Output Buffer Flag

Internal Interrupt Handling Flowchart



Software Interrupt Handling Flowchart

```
R T0 : AR ← 0 , TR ← PC   \\ NOTE TRANSFER REG.
R T1 : M[AR] ← TR , PC ← 0
R T2 : PC ← PC + 1 , IEN ← 0 , R ← 0 , SC ← 0
```

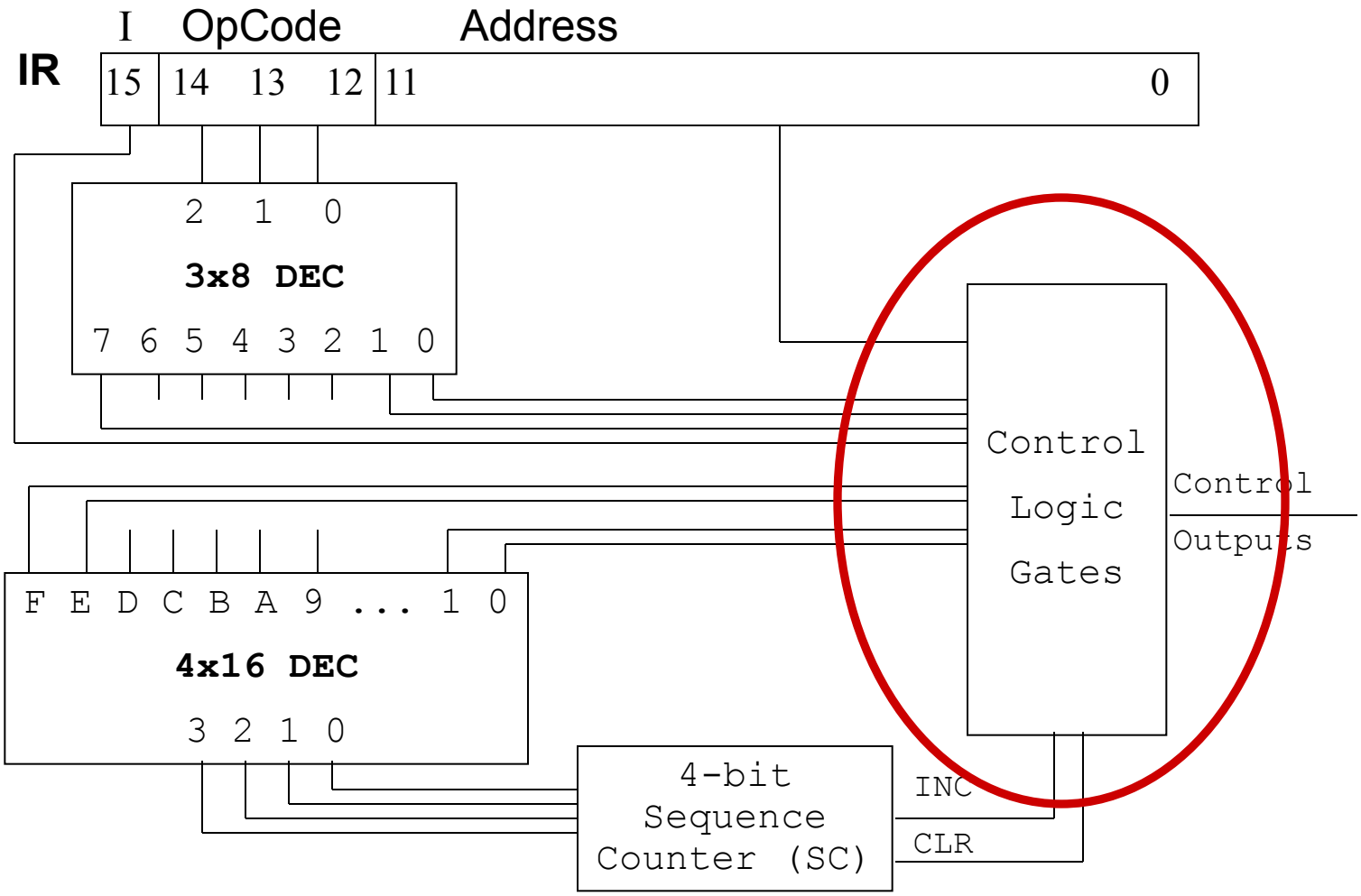


Control Logic

- Timing of microoperations requires explicit enabling of logic circuits through the Control Unit logic gates

Timing and Control - Revisited

- The ***instruction cycle*** consists of a **controlled** sequence of microoperations using control logic gates and a sequence counter.



Control Logic

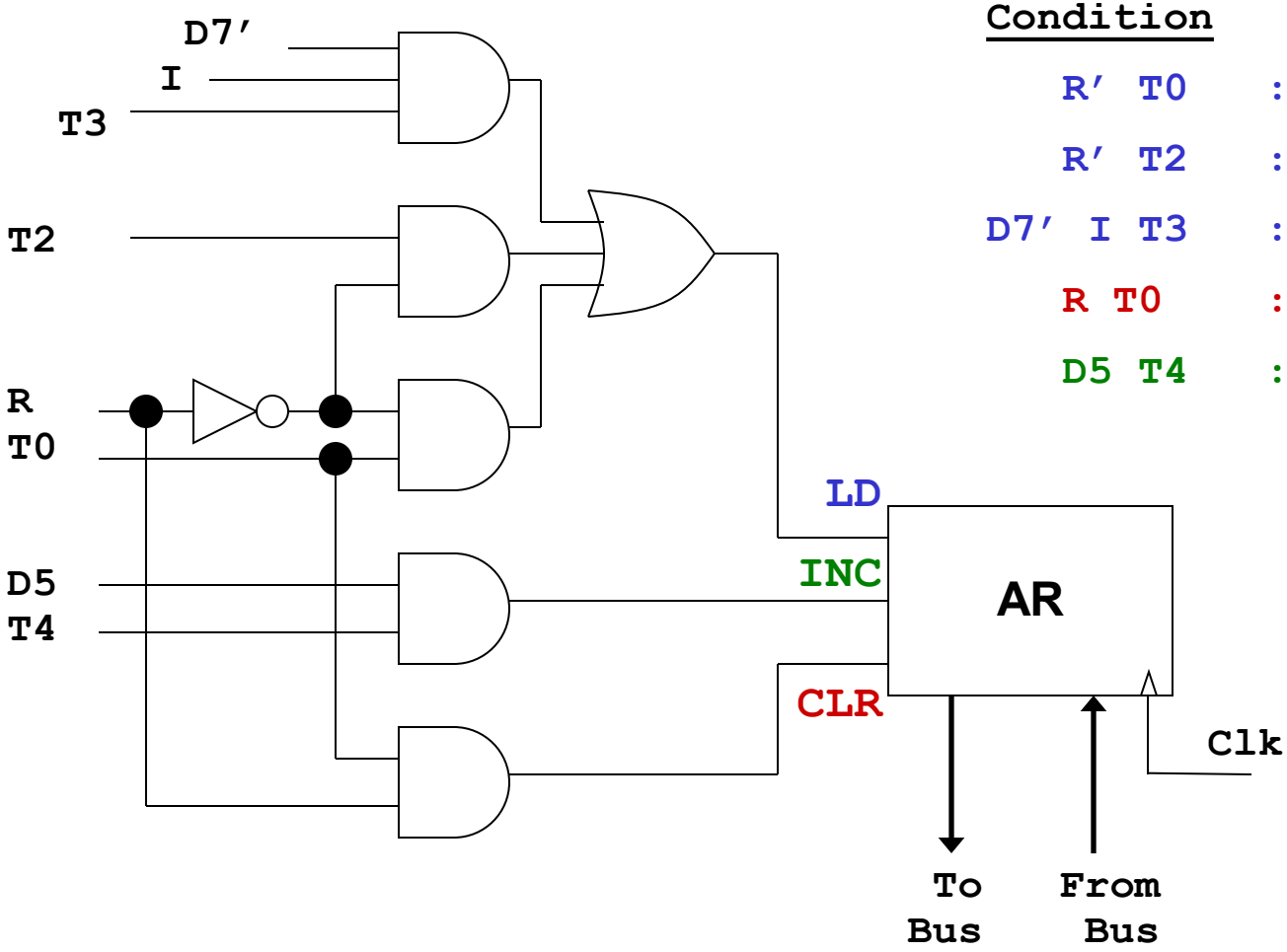
- Timing of microoperations requires explicit enabling of logic circuits through the Control Unit logic gates

- Need signals …

  - to control the inputs of the 9 registers
  - to control the read and write inputs of memory
  - to set, clear or complement the flip-flops
  - for Selector inputs to select a register for the bus
  - to control the AC adder, logic and shift circuit

Control Logic – Example: AR

- Consider all instructions that modify the AR register
- Construct an AR control circuit



Condition		Microoperation
R′ T0	:	AR ← PC
R′ T2	:	AR ← IR(0-11)
D7′ I T3	:	AR ← M[AR]
R T0	:	AR ← 0
D5 T4	:	AR ← AR + 1

Control Logic

- In previous lectures we discussed complex register circuits
  - Control was exerted through enabling inputs
- The textbook provides additional examples of controlling logic circuits
  - Control of single flip-flops
  - Control of common bus
- All registers can be adapted to controls
- Indeed, all computer circuits can be adapted to controls
  - Using enable inputs
  - Using control circuits
  - Using counters, and so on.

Representing a Complete Architecture

- Small Scale Integration
  - Logic Gates
  - Simple Circuits – Combinational & Sequential
- Medium Scale Integration
  - Functional Circuits and Control Logic
  - Arithmetic, Logic, Shift, Comparison
  - Decoders
  - Multiplexers
- Large Scale Integration
  - CPU
    - Instruction set – Arithmetic & Logic Unit Subsystem
    - Control logic – Control Unit Subsystem
    - Registers
  - Memory
    - Memory storage cells
    - Address, Data and Control Bus structure
  - Input-Output
    - Buffer registers, Control

Representing a Complete Architecture

- Hardware
  - CPU
    - Instruction set – Arithmetic & Logic Unit Subsystem
    - Control logic – Control Unit Subsystem
    - Registers
  - Memory
    - Memory storage cells
    - Address, Data and Control Bus structure
  - Input-Output, Bus Network
    - Buses, Buffer registers, Control

Memory

4096 words

16 bits/word

CPU

9 Registers
- AR, PC, DR, AC, IR, TR, OUTR, INPR, SC

7 Flip-flops
- I, S, E, R, IEN, FGI, FGO

2 Decoders
- 3x8 Operation DEC, 4x16 Timing DEC

Control logic, ALU circuits

Buses

Address

Data

Control

Buffer connections

Summary

- We adopted and discussed M. Mano's logical model of computer architecture.

- Instruction Set
- Control architecture
- Instruction processing
  - Register access
  - Memory access (Direct, Indirect)
  - I/O access
  - Interrupts
- Timing sequences of micro-operations

- In the final sequence of lectures slides we discuss some issues related to programming
  - Assembly language
  - Assembler translators
  - Programming examples