CEG2136
Computer Architecture I
Basic Computer Organization and Design

uOttawa

L'Université canadienne
Canada's university

Voicu Groza
SITE Hall, Room 5017
562 5800 ext. 2159
VGroza@uOttawa.ca

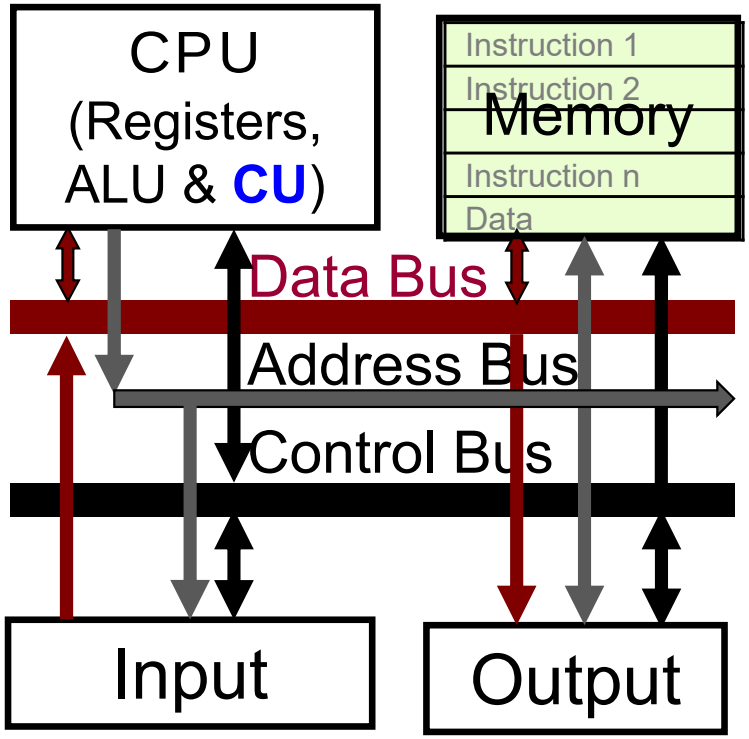
Université d'Ottawa | University of Ottawa

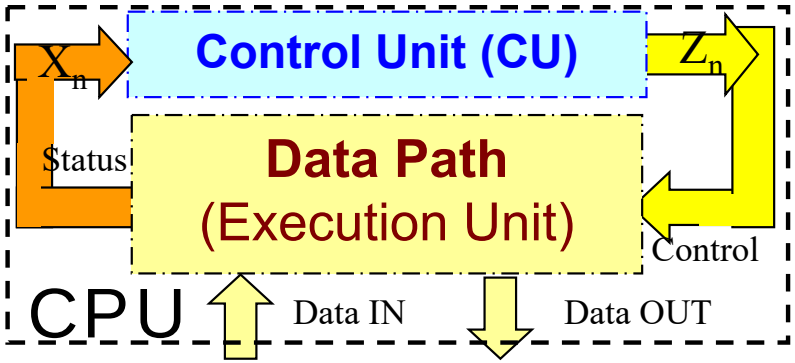
www.uOttawa.ca

Outline

- Problem specification:
  - Instruction Codes of the Basic Computer
  - Addressing Modes
  - The Datapath of the Basic Computer
  - Basic Computer Instructions
    - Register - Reference Instructions
    - Memory-Reference Instructions
    - Input-Output and Interrupt
  - Requirements for basic computer Control Unit (CU)
- ASM = tool (methodology) to solve the problem
- Design of the CU of the Basic Computer

Basic Operation of a Computer

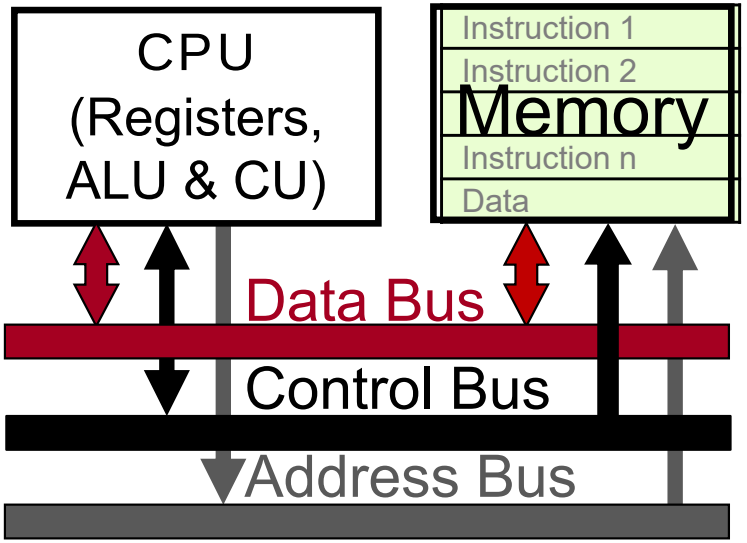




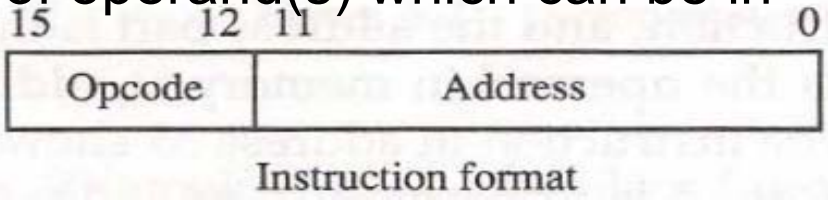
1. The computer *accepts* external information in the form of sequence of **instructions** (programs) and **data** through an input unit and *stores* it in memory
2. The *data* stored in memory is *fetched*, under *program* control, and *processed* in an ALU.
3. The processed data *leaves* the computer through an *output unit*
4. All activities inside the computer are *directed* by the Control Unit (CU).

The **Control Unit (CU)** of the CPU is a "***programmable***" *sequential circuit* which changes its transition function depending on the instruction to be performed.

Instruction Codes



- An **instruction code** is a group of bits (binary code) that instruct the computer to perform a sequence of micro-operations.
- Instruction codes together with data (operands) are stored in the computer memory.
- The control unit (CU) then interprets the binary code of the instruction and proceeds to execute it by performing a sequence of micro-operations.
- An instruction code is usually divided in two parts: an **operation code (opcode)** and an **address** code.
- The **opcode** (operation code) defines the operation to be performed: addition, subtraction, logic AND, etc.
- The **address** code specifies the address(es) of operand(s) which can be in
  - memory, for *Memory Reference Instructions* (MRI) or in
  - registers, for *Register Reference Instructions* (RRI) or *I/O Reference Instructions* (IOI).



Instruction format

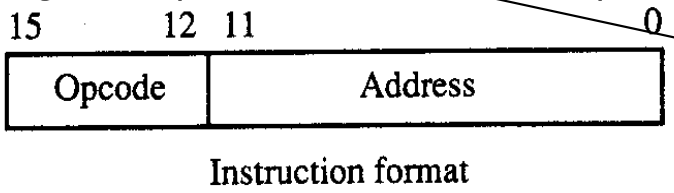
HERE: operands' sources: one in memory at specified address and the other one in accumulator; result destination = accumulator.

Instruction Codes

- The control unit receives the instruction code from the memory, interprets the operation code (opcode) and then issues a sequence of control signals to initiate the necessary sequence of micro-operations to be performed on the specified operands.
- An instruction code must specify not only the operation (defined by the operation code), but also the *registers* or *memory* locations where the operands are to be found, as well as the *register* or *memory* location where the result is to be stored.
- A memory location can be specified within the instruction code by specifying its *address* in memory.
- A processor register can be specified within the instruction code by assigning k bits that identify one of the 2^k possible processor registers available in the system.
- In this chapter, we will discuss the basic organization, functionality, and design of a small-scale computer system.

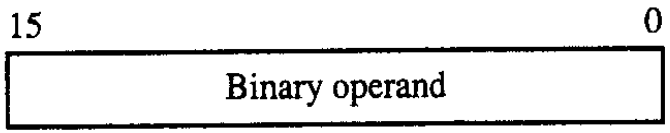
Stored Program Organization

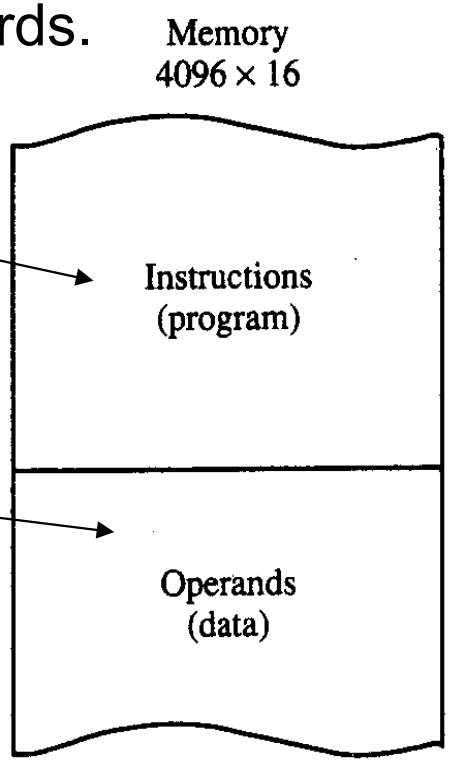
- A 4096 × 16 memory unit is used here ⇒ 4096 × 16-bit words.
- The memory is divided into two parts:
  - A program part, storing the program (set of instructions) to be executed,



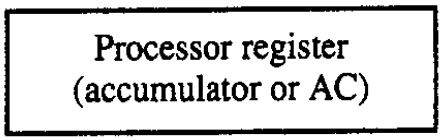
Instruction format

  - A data part, storing the data (operands) to be processed by operations encoded in the instructions of the program.





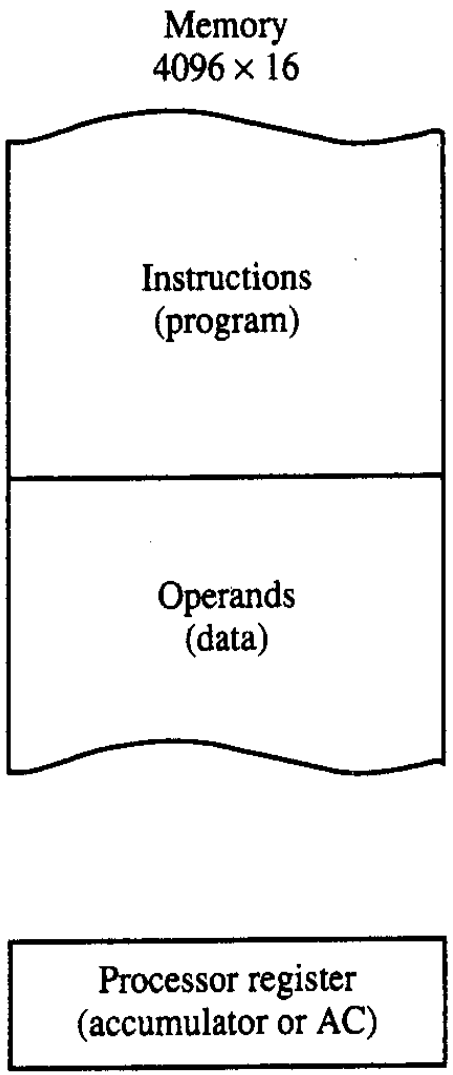
- Each **instruction** in the program part of the memory is 1-word long (16 bits) and it is divided in two parts:
  - The first part is composed of 4 bits (bits 12 to 15) which specifies the operation code (**opcode**) of one of a maximum of $2^4 = 16$ possible operations that can be performed.
  - The second part is composed of 12 bits (bits 0 to 11) and it specifies the **address of the operand** in memory (MRI) or may encode other operations which do not require access to the memory (RRI or IOI).



- The 12 bits in the address part of the instruction code can specify maximum of $2^{12} = 4096$ different word addresses in the memory.

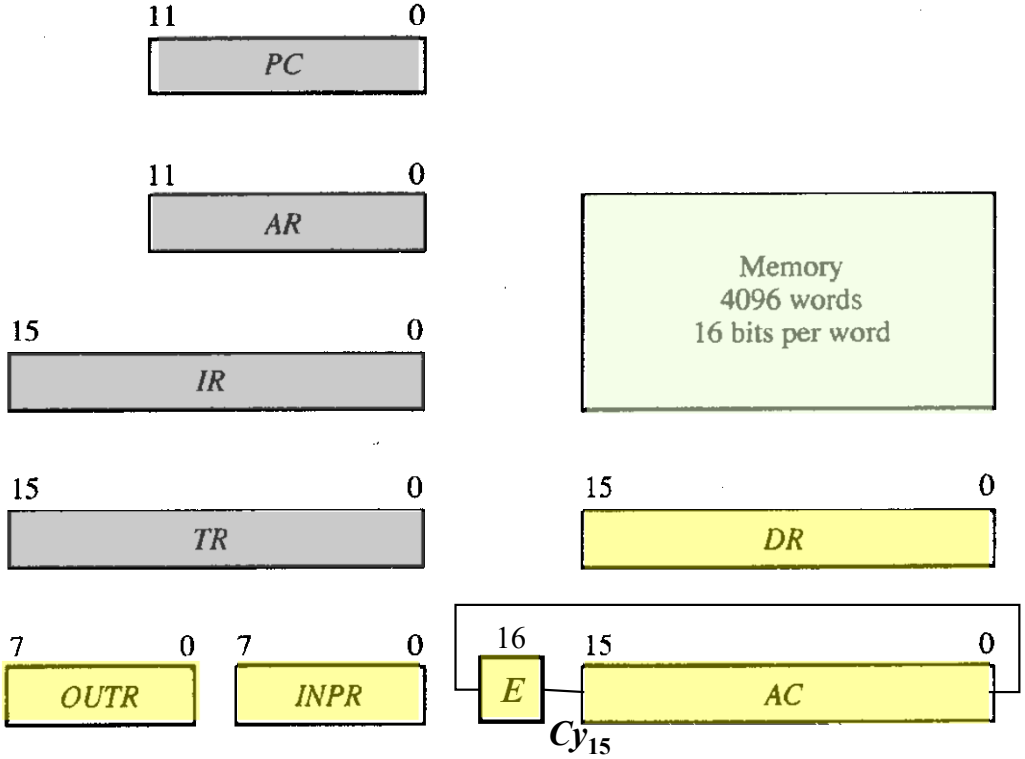
Stored Program Operation

- Each word in the data part of the memory holds a 16-bit operand.
- The accumulator *AC* is a processor register that is used here to store one operand of the operation to be performed.
- E.g., for an addition: operand 1 + operand 2 = result ;
  operand 1 is stored in memory at the address given in the instruction, while operand 2 is in AC; result will go to AC
- A program instruction is executed in the following sequential order:
  1. The control unit reads the 16-bit instruction code from the program portion of the memory.
  2. The 12-bit address part of the instruction code is then used to read the 16-bit operand from the data portion of the memory.
  3. The 4-bit operation code is then used to perform the desired operation on the operand just read.

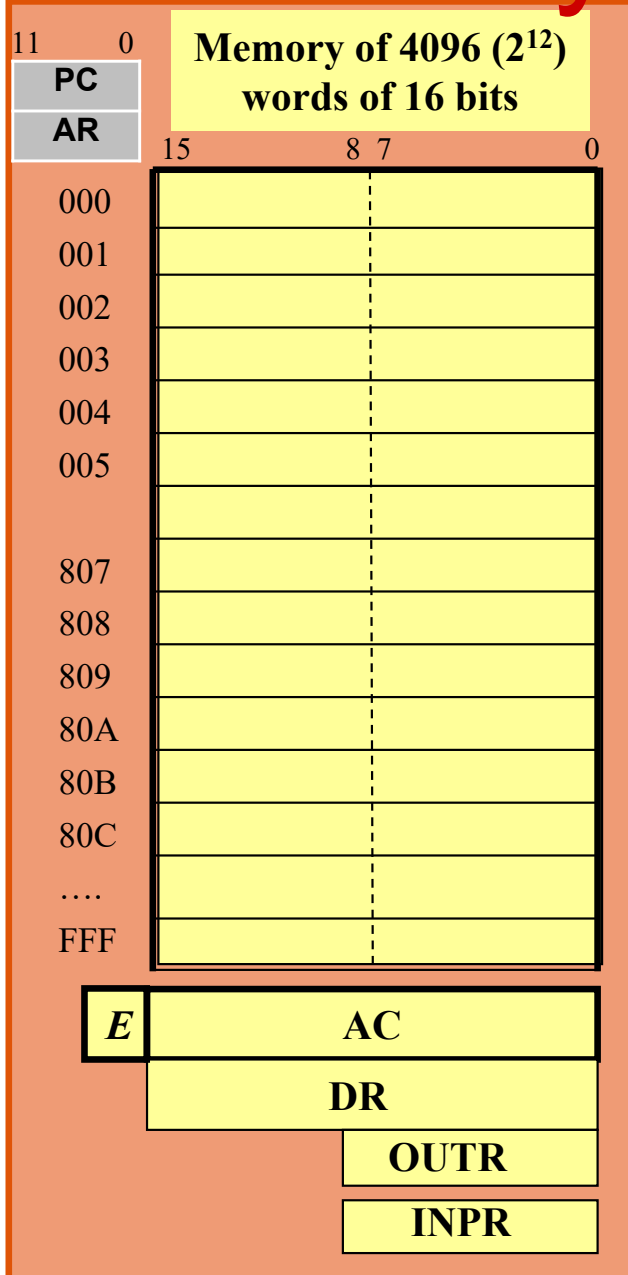


uOttawa

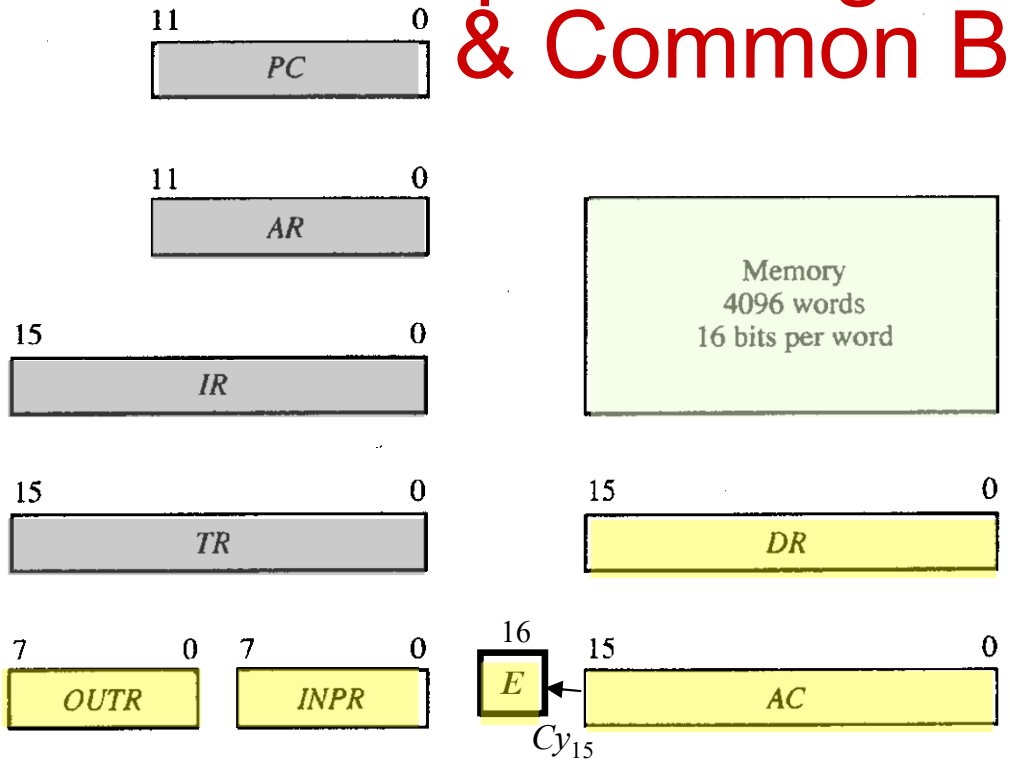
Basic Computer Registers & Memory

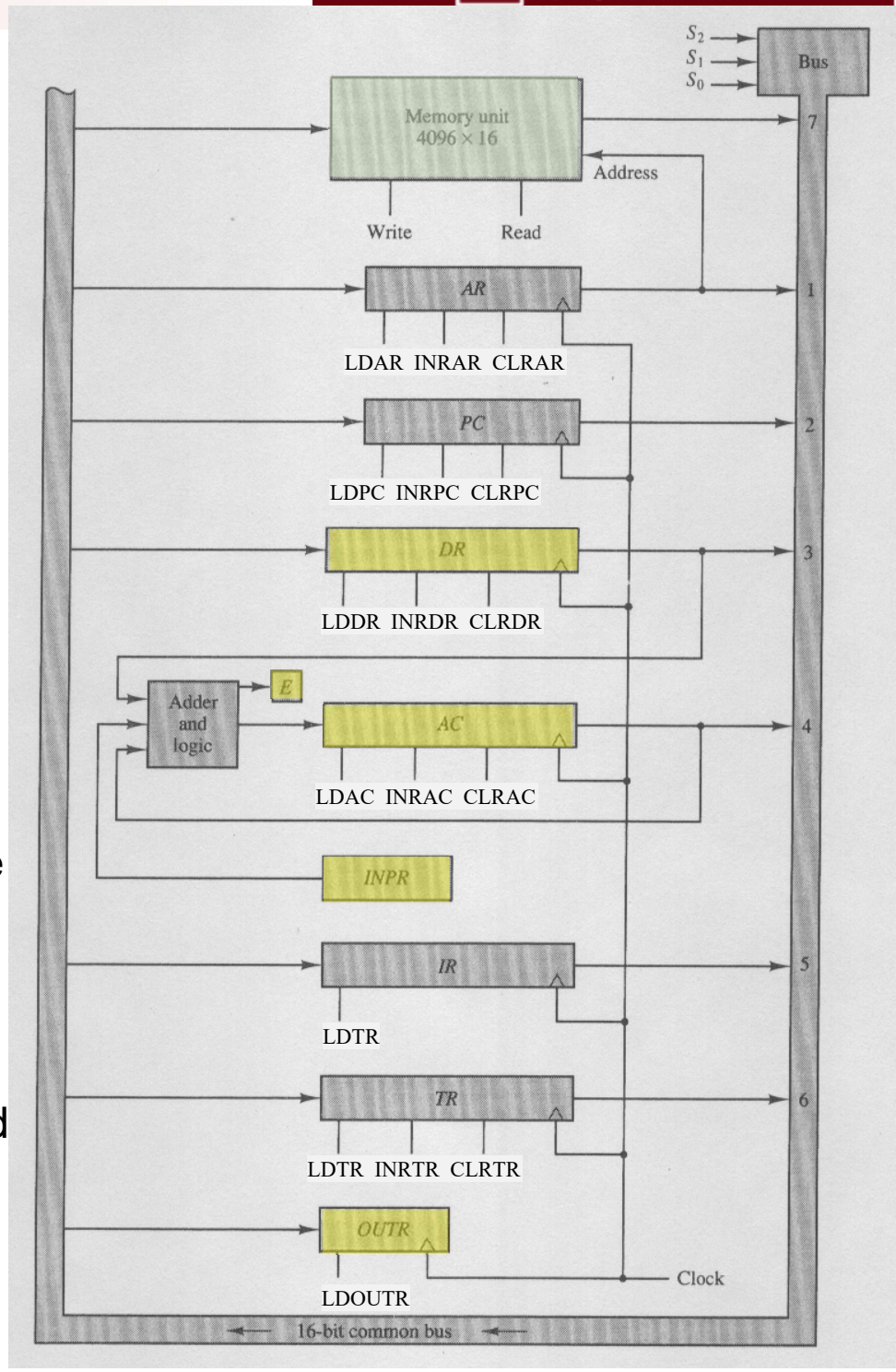


Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
TR	16	Temporary register	Holds temporary data
AR	12	Address register	Holds address for memory
PC	12	Program counter	Holds address of instruction
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character



Basic Computer Registers & Common Bus





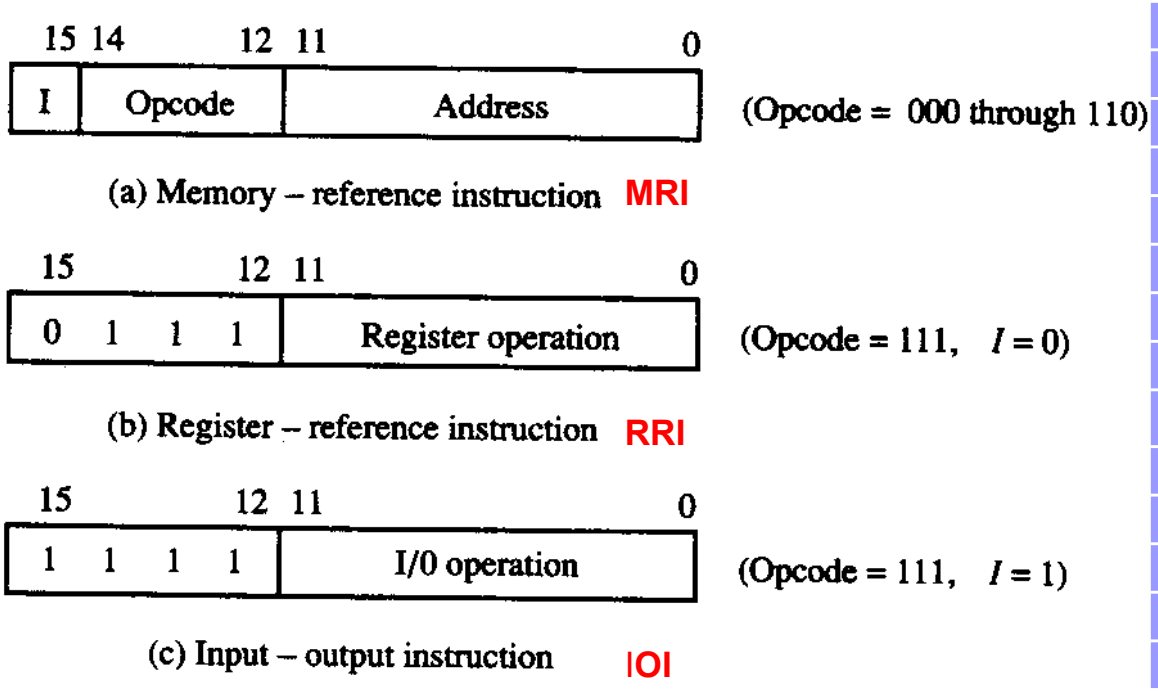
- The content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle.
- The clock transition at the end of the cycle transfers the content of the bus into the designated destination register and the output of the adder and logic circuit into AC.
- For example, the two microoperations

  DR ← AC and AC ← DR

  can be executed at the same, time.

Basic Computer Instruction List (Hex)

There are three types of instructions which have different formats:



Symbol	I=0	I= 1	Description
MRI			
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
RRI			
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Halt computer
IOI			
INP		F800	Input character to AC
UT		F400	Output character from AC
SKI		F200	Skip on input flag
SKO		F100	Skip on output flag
ION		F080	Interrupt on
IOP		F040	Interrupt off

Addressing Modes

- Register Addressing (RRI / IOI)
  - Operands reference internal registers
  - Also called *inherent addressing* (Motorola)
- Immediate Addressing (not implemented here)
  - Operand is constant and is contained in the instruction word *immediately* following the opcode
- Direct Addressing (MRI)
  - The Operand is to be fetched from the memory location whose address is given by the 12 bits following the opcode in the instruction word
- Indirect Addressing (MRI)
  - Instruction specifies where the **address of the operand** is located
- Indexed addressing (not implemented here)
  - Finds the data address using an index (an offset)
- Relative Addressing (not implemented here)
  - Add offset to current value of the PC

Notation used in Comments

- Register Name: Indicates a register and its contents
  - Example: AC refers to the contents of accumulator AC
- Right arrow (→) indicates data transfer operation, while ↔ indicates an exchange of data
  - Example: AC → DR indicates that the contents of AC are transferred (copied) to DR
- M[…] Contents of a memory location
  - Example: M[1234**H**] → AC indicates that the contents of memory location **HEX**1234 are transferred to AC (also: M[**$**1234] → AC)
- M[M[…]] *Indirect* addressing – inner parentheses specifies a memory address that contains the data address
  - Example: AC → M[M[1234H]] indicates that the contents of AC are transferred to a memory location whose address is found in memory location with address 1234H (syntax: STA @1234H)

Inherent Addressing (Register Addressing)

RRI / IOI

- All data for instructions are in the CPU
- Instructions with inherent addressing mode are:
  - among the fastest to execute
  - coded with the least amount of bits.
- This mode is also called Register Addressing or Implied Addressing
- All RRI and IOI of the Basic Computer employ the Inherent Addressing mode

CLA ; 0 → AC

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Opcode														
CLA =7800_H	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

INC ; AC ← AC+1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Opcode														
INC=7020_H	0	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0

Immediate Addressing

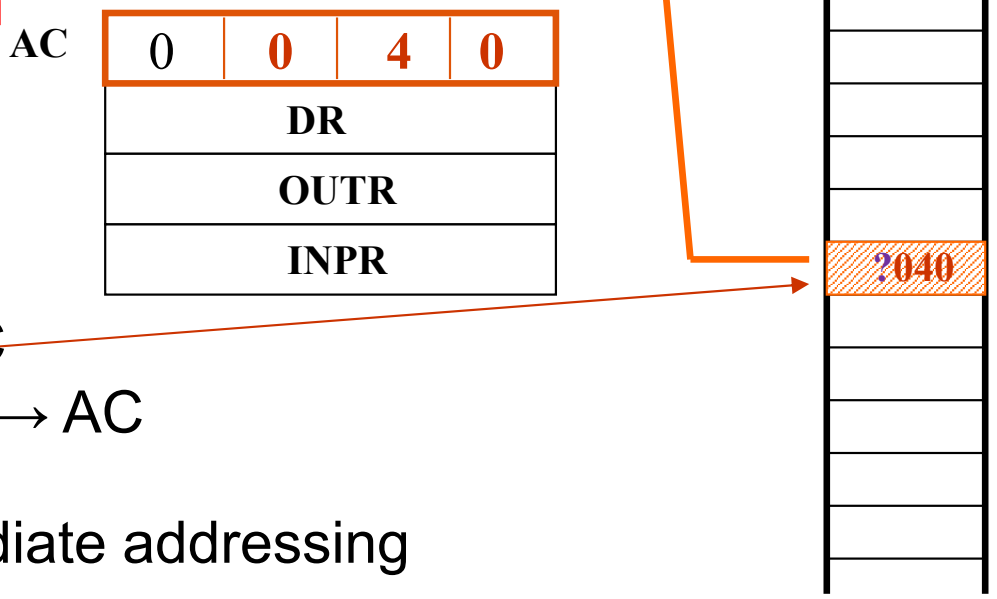
not implemented here

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Opcode				Operand Value											
LDA #40H	?	?	?	?	0	0	0	0	0	1	0	0	0	0	0	0

An argument is contained in the bits immediately following the opcode.

It is not implemented in this BASIC COMPUTER !!! So the opcode is marked with "**?**" in the following example.

The lsb 12-bit contents of the instruction word (i.e., the operand) are moved to accumulator AC



Immediate Addressing Examples

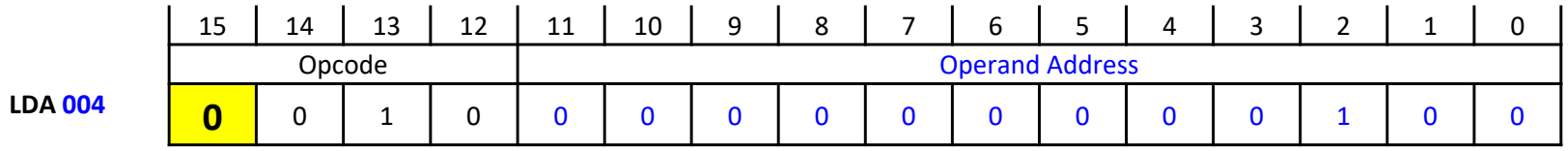
```
LDA   #64     ; Decimal 64 → AC
LDA   #64H    ; Hexadecimal 64 → AC
```

- The # symbol indicates immediate addressing
- It is easy to forget the #

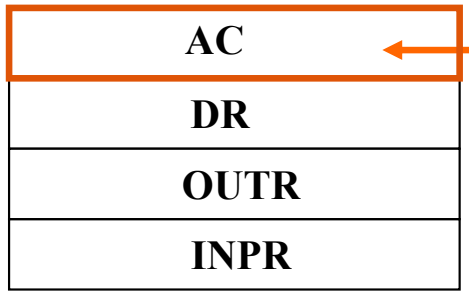
Direct Addressing

MRI

- Instruction contains the data address
  - Single-level addressing
  - Operand's address is given by the low-order 12 bits following the opcode.
  - Instructions encoded with opcodes 0 - 6 of the *Instruction List* of the Table on slide 10 (with **msb** IR_{15}**=0**) access addresses $000–$FFF directly

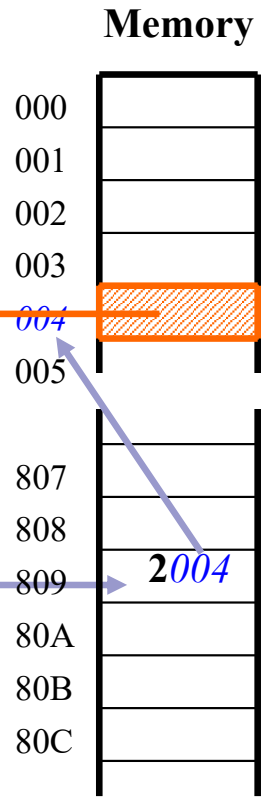


Example: **LDA 4** ;A ← M[4]



2. The 16-bit contents of the selected memory location are copied, as specified by the instruction, to the accumulator AC.

1. The lsb 12-bit of the instruction word (*004*) represent the address of the memory location where the operand is stored



Indirect Addressing

MRI

- Also called *pointer addressing*
- Instruction contains address of memory location (marked with "I" or "@"or …) that holds the data address (*pointer*)
- Two level addressing mechanism
  - First level provided by instruction gives *address of memory containing an address*
  - Second level is the address that specifies where the data is located
  - Instructions encoded with opcodes 8 - E (msb IR_{15}=1) access addresses $000–$FFF indirectly. Actually the memory space they can address is 64kW (kilo words)

Example:

```
LDA  9BA  I
  ;AC ← M[M[9BAH]]
```

2. The 16-bit contents of memory location $9B5 are moved to the accumulator AC

1. The lsb 12 bits of the instruction word (*$9BA)* represent the *address* of the memory location where **data** address (9B5) is stored; **data** address points to the memory location where **data** is stored

AC	1234
DR	
OUTR	
INPR	

Memory	
9B1	A*9BA*
9B5	1234
9B6	
9B7	
9B8	
9B9	
9BA	09B5
9BB	

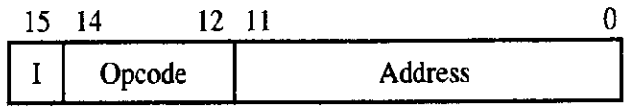
Direct VS Indirect Address

The instruction code is composed of a 3-bit opcode, a 12-bit address, and an indirect address mode bit *I* (*I* = 0 for a direct address, and *I* = 1 for an indirect one).

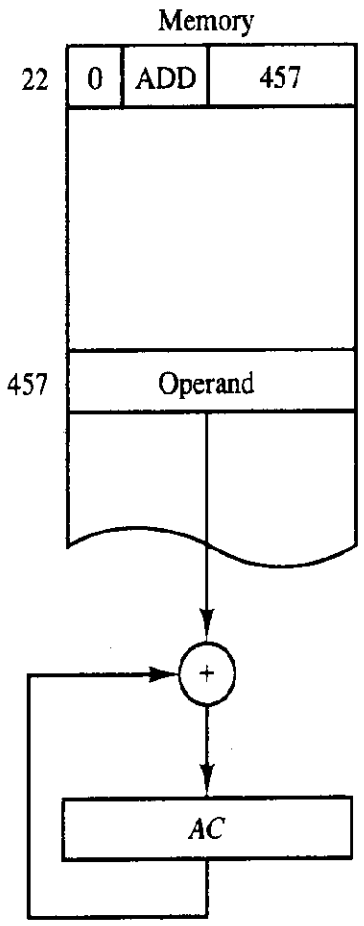
Fig. (b) shows a direct address mode (*I* = 0) with an opcode of an ADD operation. The 12-bit address code holds the binary equivalent of 457. ⇒ The operand is found in address 457 of the memory.

The operand is fetched and added to the number stored in the accumulator *AC.*

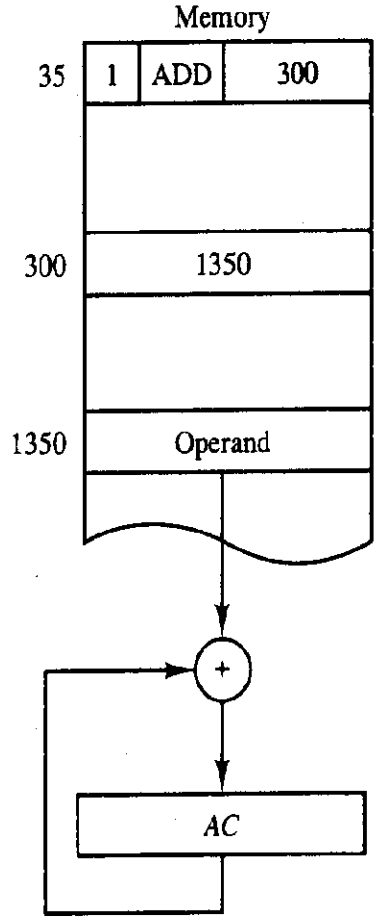
Fig. (c) shows an indirect address mode (*I* = 1) with an opcode of an ADD operation. The 12-bit address code holds the binary equivalent of 300. ⇒ The address of the operand is fetched from address 300, which contains the binary equivalent of 1350. The operand is then fetched from address 1350 and added to the number stored in the accumulator *AC* .



(a) Instruction format



(b) Direct address

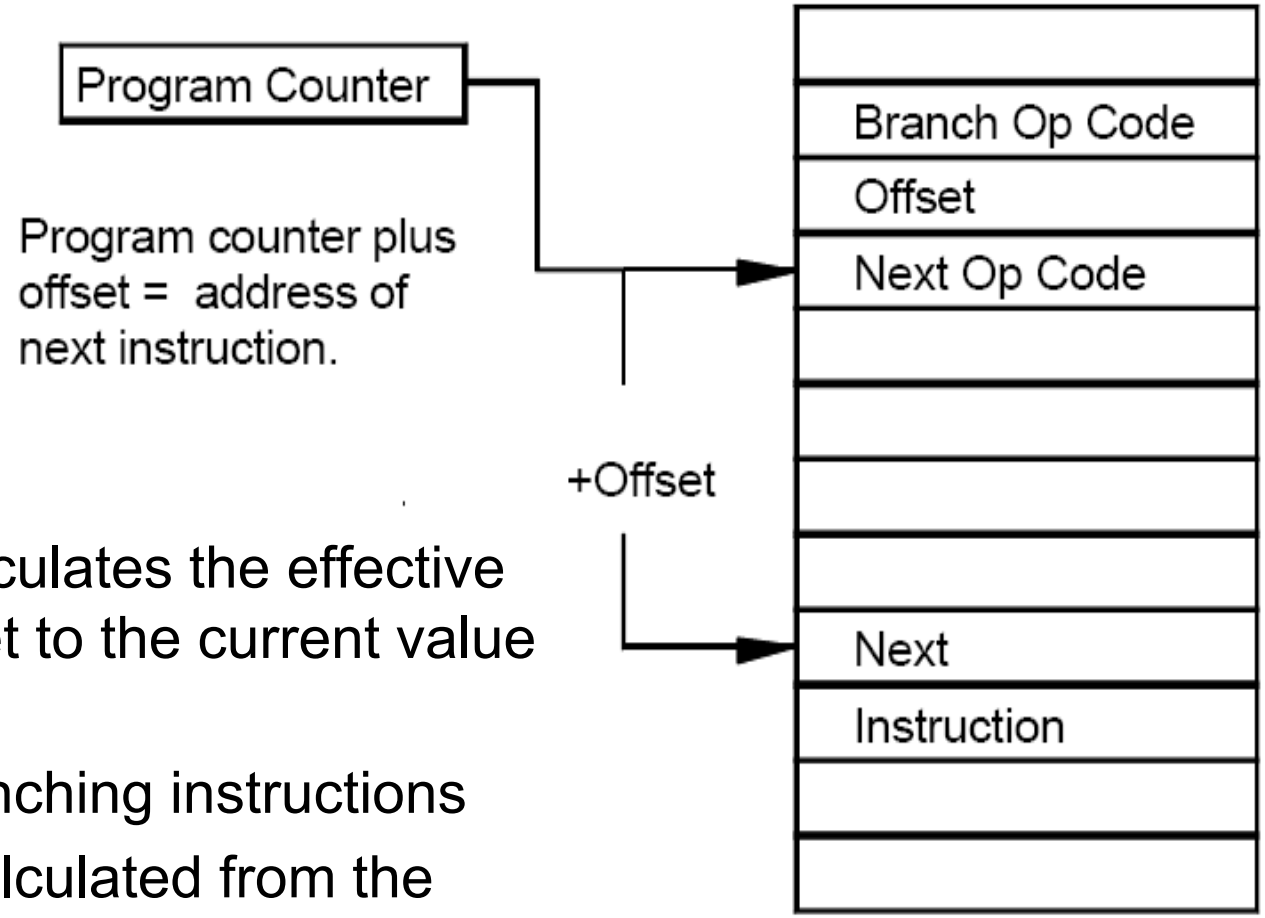


(c) Indirect address

Relative Addressing

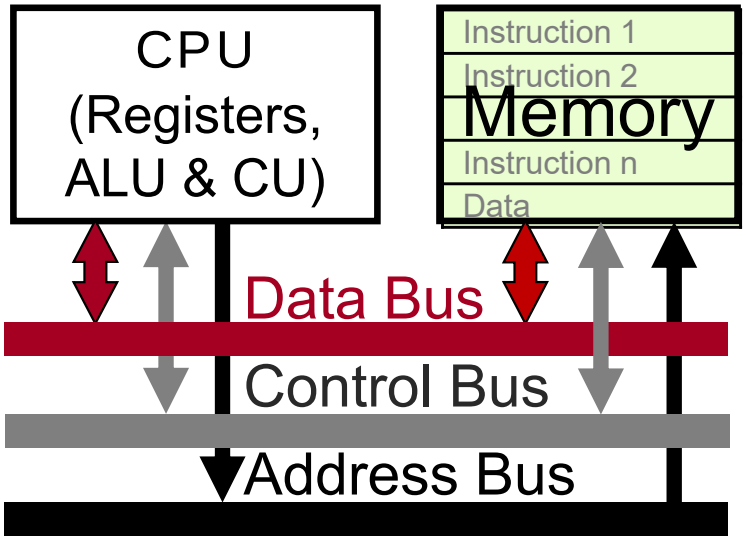
not implemented here

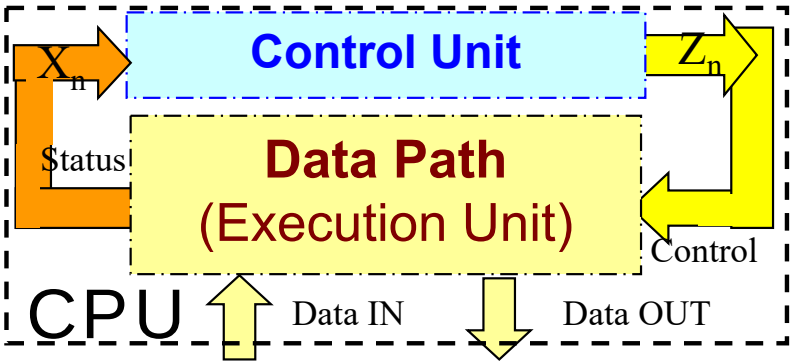
- The address of the operand is called the **effective address**.
- For instance, the effective address in the previous Fig. (b) is 457, whereas it is 1350 in Fig. (c).



- Relative addressing calculates the effective address by adding offset to the current value of the PC
  - Used mostly for branching instructions
  - Normally offset is calculated from the address of the next op-code
- Not implemented in our BASIC Computer

Basic Computer



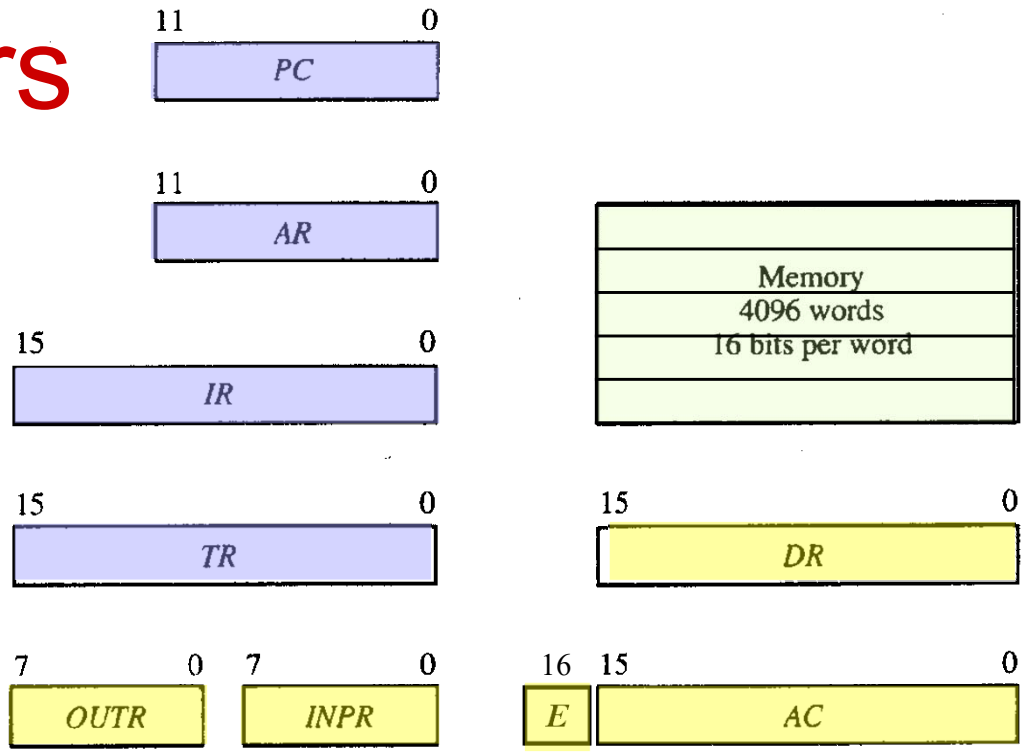


The CPU's main components are **Data Path** and **Control Unit**.

The **Data Path** consists in *Registers* and *ALU*

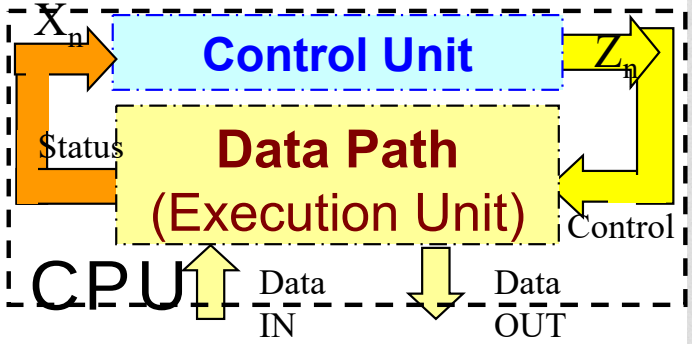
The **Control Unit (CU)** is a "***programmable***" *sequential circuit* which changes its transition function depending on the instruction to be performed.

Computer Registers

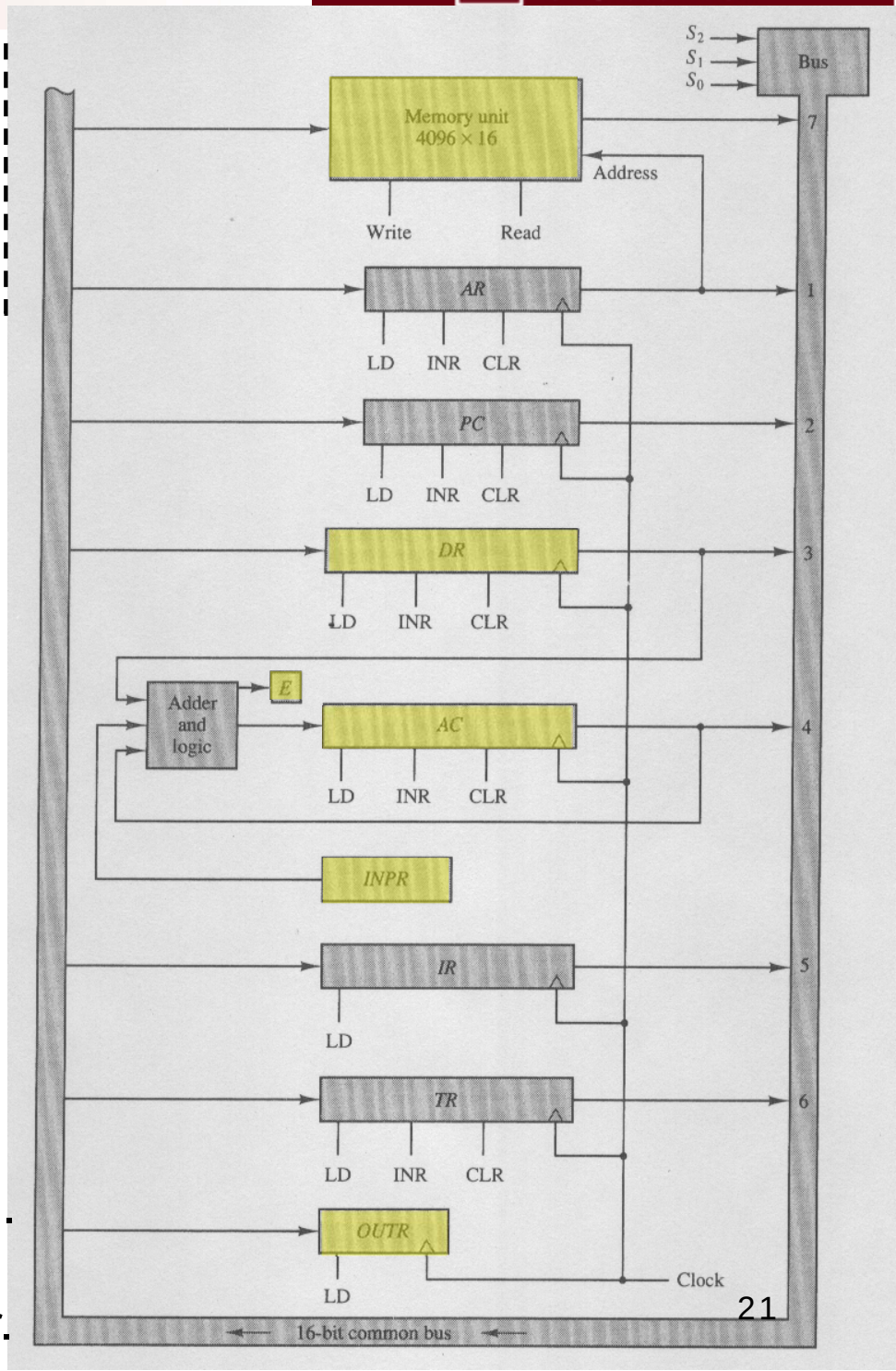


- The memory unit has a capacity of 4096 words with 16 bits each.
- As shown in Figure 29(a), each 16-bit instruction code is composed of 12 address bits, a 3-bit opcode, and one bit to specify a direct or indirect address.
- The data register (*DR*) holds the operar read from the memory.
- The accumulator (*AC*) is a general purpose processing register.
- The instruction register (*IR*) holds the instruction read from the memory.
- The temporary register (*TR*) holds temporary data during processing
- The address register (*AR*) is a 12-bit register that holds the address of the word to be accessed in the memory.
- The program counter (*PC*) is a 12-bit register that holds the address of the next instruction to be fetched from the memory after the current instruction is executed.
- *INPR* is an input register that holds an 8-bit code of a character read from an input device. *OUTR* is an output register that holds an 8-bit code of a character to be transferred to an output device.

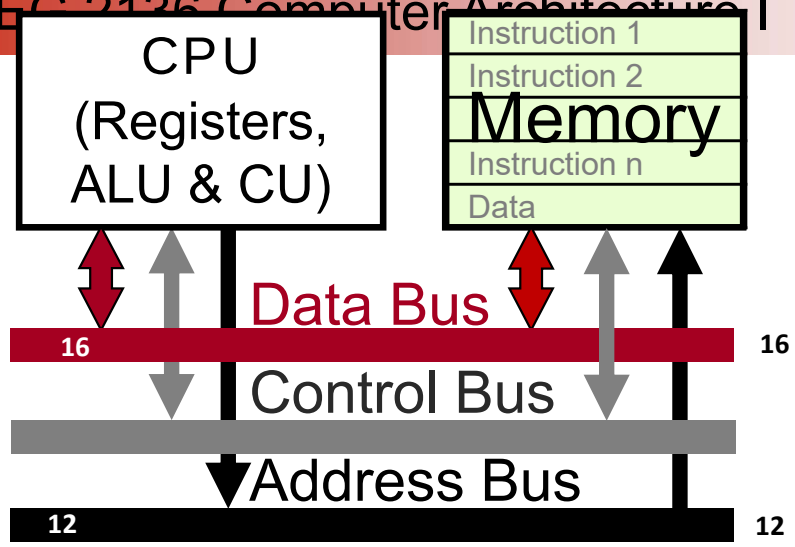
Data Path

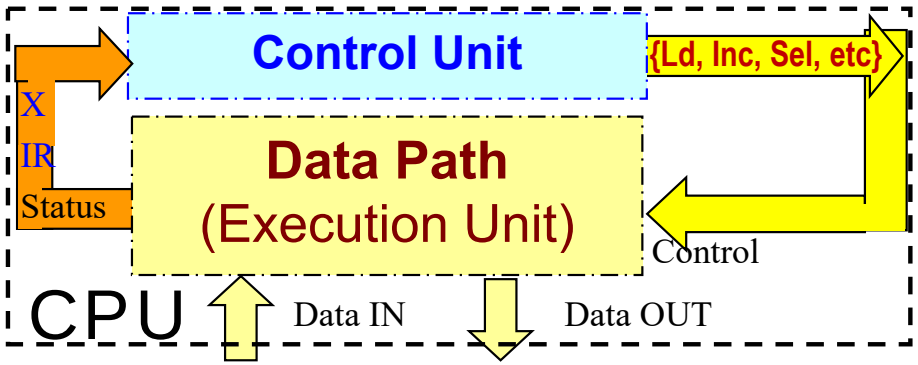


- The common bus lines are connected to the data input lines of each register as well as the data inputs of the memory.
- The particular register whose *LD* (load) input bit is enabled, receives the data from the bus during the next clock pulse.
- The content of the bus is loaded into the memory when its *Write* control bit is activated.
- The memory places one of its 16-bit words on the bus when its *Read* control bit is activated **and** $S_2S_1S_0 = 111$.
- Two of the registers connected to the bus have 12 bits only (*AR* and *PC*).
- When the content of *AR* or *PC* is applied to the 16-bit bus, the bus' 4 msb's are reset to '0'.
- When *AR* or *PC* loads the content of the bus, only the 12 lsb's are transferred to the register.

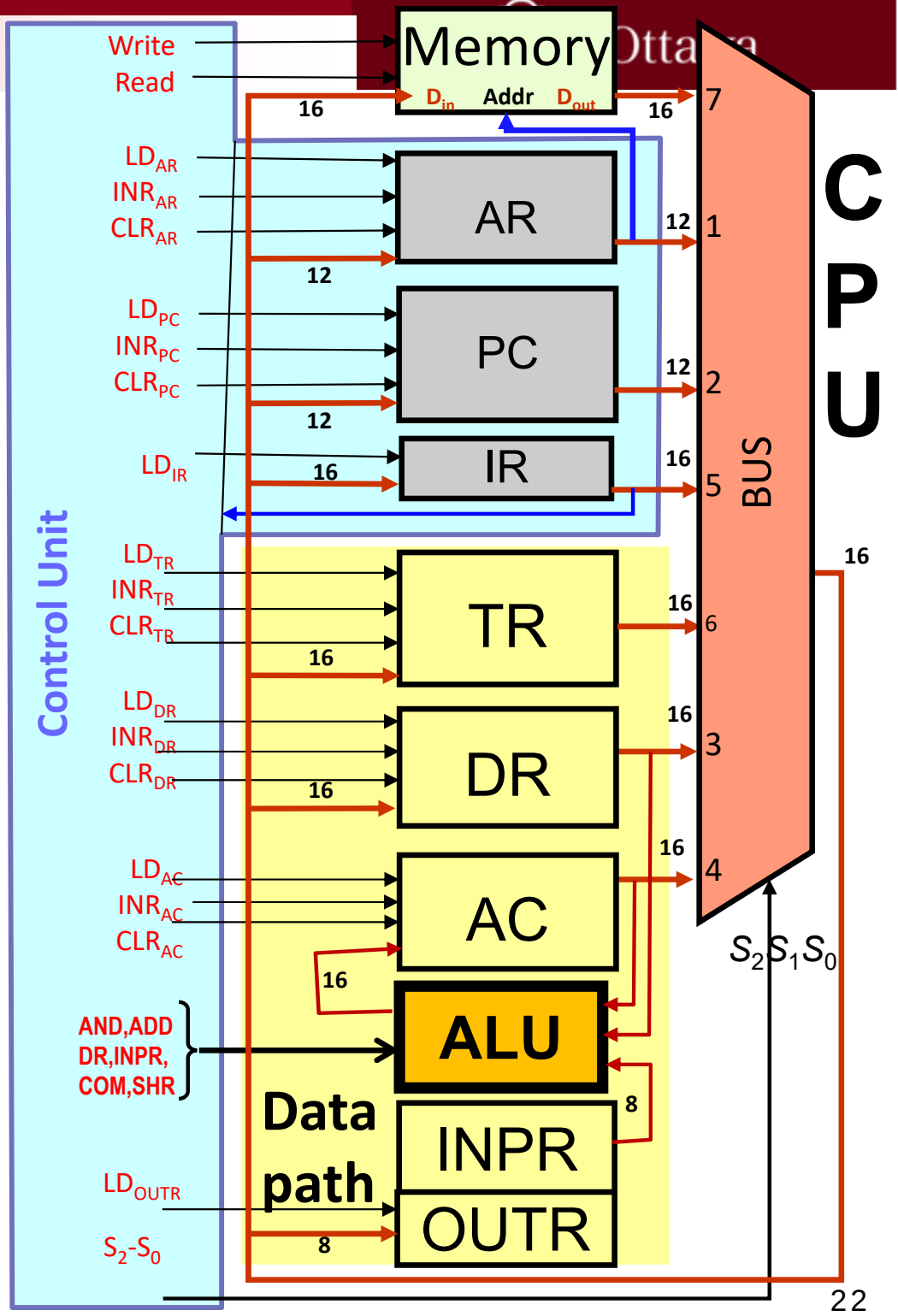


BASIC COMPUTER



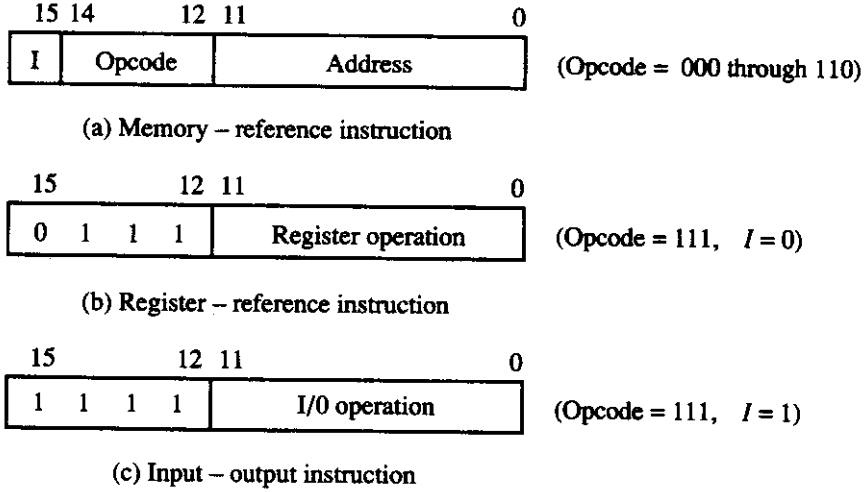


- The Basic Computer CPU has a **Data Path** (8 registers & memory unit, interconnected via a common bus) and a **Control Unit**.
- The bus selection bits S_2, S_1, and S_0, determine the output of the specific register or memory to be placed on the bus lines at any given time.
- The number along each BUS input in next figure shows the decimal equivalent of the required binary selection (S_2, S_1, and S_0).



uOttawa

Basic Computer Instructions



There are 3 instruction formats.

The opcode is composed of 3 bits ($I_{14}I_{13}I_{12}$)

000-110 = operations to be performed on *AC*

- the rest of the12-bit (I_{11} - I_0) specify an address in memory:
- I_{15} = 0 => instruction with direct address
- I_{15} = 1 => instruction with indirect address

111 & I_{15} = 0 => a register reference instr.

111 & I_{15} = 1 => an input-output instruction. the other 12 bits (I_{11} - I_0) are used to specify the type of operation to be performed

Symbol	I=0	I= 1	Description
MRI			
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
RRI			
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Halt computer
IOI			
INP		F800	Input character to AC
UT		F400	Output character from AC
SKI		F200	Skip on input flag
SKO		F100	Skip on output flag
ION		F080	Interrupt on
IOP		F040	Interrupt off

Basic Computer Instruction List (Binary)

D_7	Instr. Dec	Addressing Mode I_{15}=0	I_{15}=1	op code I_{14}	I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
		MRI Direct Addr.	**MRI** Indirect Addr.				Memory				A	D	D	R	E	S	S	
	D_0	**AND**=$0addr	AND_i =$8(addr)	0	0	0	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_1	**ADD**=$1addr	ADD_i =$9(addr)	0	0	1	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_2	**LDA** =$2addr	LDA_i =$A(addr)	0	1	0	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
D_7=0	D_3	**STA** =$3addr	STA_i =$B(addr)	0	1	1	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
mem	D_4	**BUN** =$4addr	BUN_i =$C(addr)	1	0	0	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_5	**BSA** =$5addr	BSA_i =$D(addr)	1	0	1	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_6	**ISZ** =$6addr	ISZ_i =$E(addr)	1	1	0	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
		RRI	**IOI**	**1**	**1**	**1**	C o d e				Extension							
	D_7	**CLA** =$7800	INP = $F800	1	1	1	**1**	0	0	0	0	0	0	0	0	0	0	0
	D_7	**CLE** =$7400	OUT = $F400	1	1	1	0	**1**	0	0	0	0	0	0	0	0	0	0
	D_7	**CMA** =$7200	SKI = $F200	1	1	1	0	0	**1**	0	0	0	0	0	0	0	0	0
	D_7	**CME** =$7100	SKO = $F100	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0
	D_7	**CIR** =$7080	ION = $F080	1	1	1	0	0	0	0	**1**	0	0	0	0	0	0	0
D_7=1	D_7	**CIL** =$7040	IOF = $F040	1	1	1	0	0	0	0	0	**1**	0	0	0	0	0	0
reg	D_7	**INC** =$7020	n/a	1	1	1	0	0	0	0	0	0	**1**	0	0	0	0	0
	D_7	**SPA** =$7010	n/a	1	1	1	0	0	0	0	0	0	0	**1**	0	0	0	0
	D_7	**SNA** =$7008	n/a	1	1	1	0	0	0	0	0	0	0	0	**1**	0	0	0
	D_7	**SZA** =$7004	n/a	1	1	1	0	0	0	0	0	0	0	0	0	**1**	0	0
	D_7	**SZE** =$7002	n/a	1	1	1	0	0	0	0	0	0	0	0	0	0	**1**	0
	D_7	**HLT** =$7001	n/a	1	1	1	0	0	0	0	0	0	0	0	0	0	0	**1**

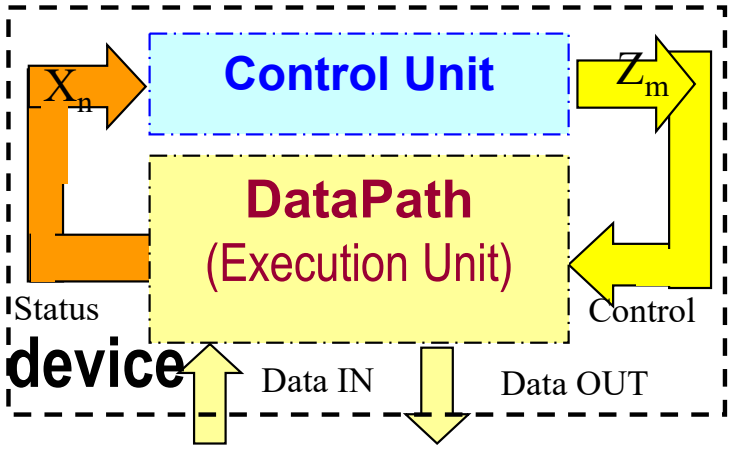
Binary encoding **One-hot** (*bit-per-state*) encoding "addr" = 12 bit address of the operand
"(addr)" = address of the operand address

Instruction Set Completeness

- A computer instruction set is complete if it can be used to evaluate any function that is known to be computable.
- To be complete, the instruction set must contain enough instructions in each of the following categories:
  1. Arithmetic, logic, and shift instructions.
  2. Instructions for moving information to and from memory and processor registers.
  3. Program control instructions together with instructions that check status conditions.
  4. Input and output instructions
- Program control instructions, such as branch instructions, are used to change the sequence in which the program is executed.
- Input and output instructions are needed for communication between the computer and the user (outside world).
- In this context, the instruction list of the Basic Computer is complete as it provides enough instructions in each category.

Algorithmic State Machine (ASM)

- The ASM method is utilized to solve a given problem, assuming that the target is a digital electronic **device,** consisting in a **datapath** and a **control unit**.



- The following are the five main steps in the ASM methodology:

ASM # 1. Using ***pseudocode*** describe the algorithm to be executed by the device

ASM # 2. Convert the pseudocode into an ***ASM flowchart* with *RTL***

ASM # 3. Design the ***datapath*** based on the ASM flowchart

ASM # 4. Create an ***ASM detailed chart*** (equivalent to FSM) with *control signals* that have to be generated by the *control unit* to direct the *datapath*

ASM # ***5.*** ***Logic Synthesis*** of the *control unit circuits* based on the detailed ASM chart

- If followed correctly, the ASM method produces a hardware design in a systematic and logical manner
  - Very robust and easily modified
  - Refined through pseudocode iterations

Rules for a Safe Design

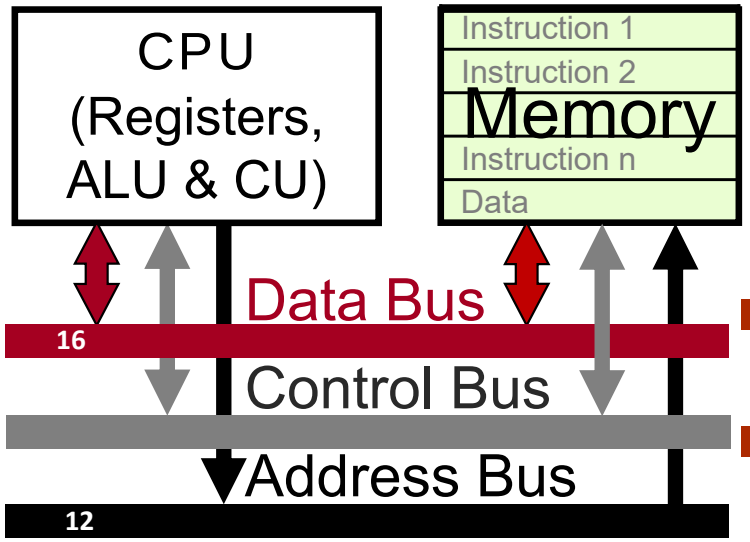
1. Keep the design simple and modular
   - Use an iterative procedure, if necessary, to build through refinement
   - Implement one module at a time, while unit testing each one
2. Develop good documentation during the design
3. Beware of clock skew
   - Use similar path lengths to all flip-flops
   - Do not use gated clocks!
   - Use all positive-edge-triggered or all negative-edge-triggered flip-flops
4. Be wary of asynchronous inputs to the circuit
   - Avoid them whenever possible!
   - Synchronize any ones that cannot be avoided
   - Use debounced switches to provide clean input signals
5. Beware of noise on power and signal lines
6. Avoid dependencies on minimum logic gate delays
7. Initialize all flip-flops to known values at the beginning

BASIC COMPUTER INSTRUCTION CYCLE

RTL ASM specifications and general design

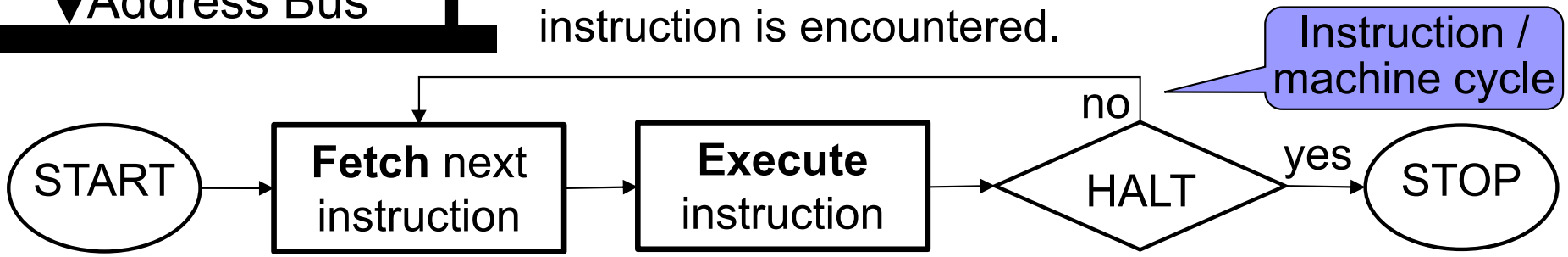
Basic Instruction Cycle

- A program residing in the memory of the computer consists of a sequence of instructions. Before running the program, it is loaded (somehow!?) onto the Memory. When run a program, each instruction is executed in several steps:



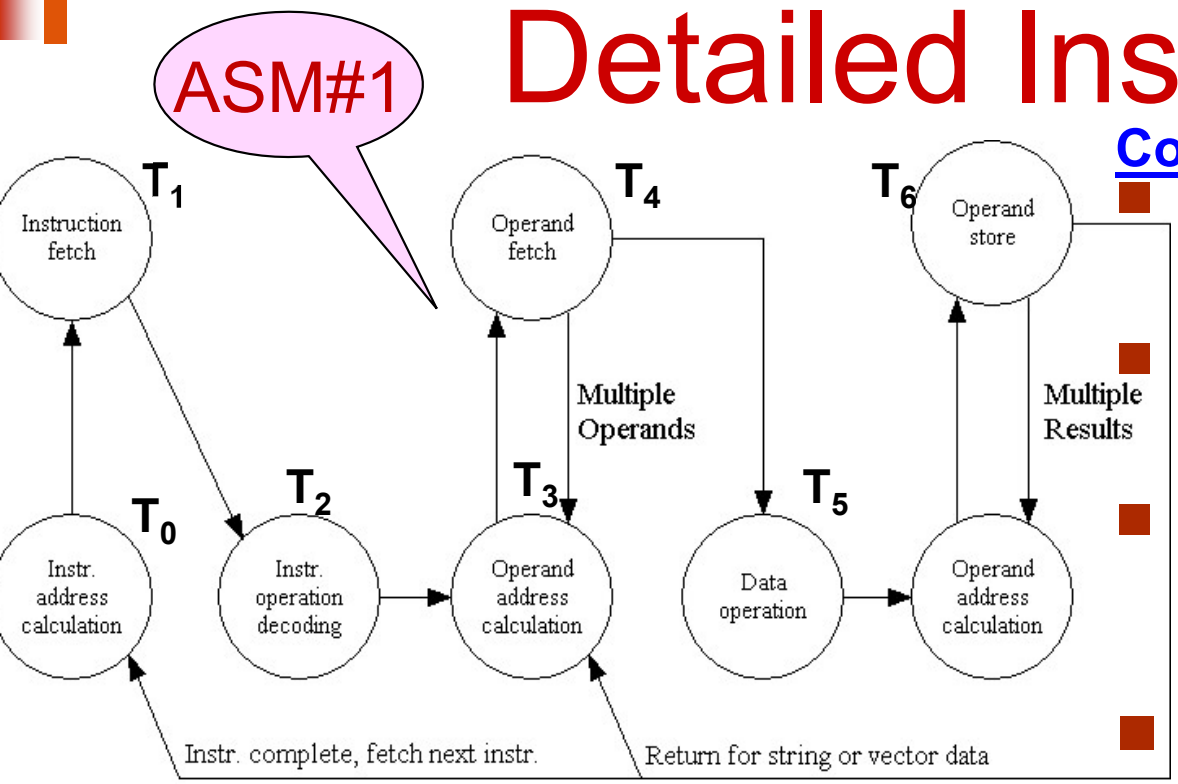
1. **Fetch an instruction from memory**.
   Read operand's **effective address** from memory **if** the instruction has an *indirect address*.
2. Decode the instruction.
   Execute the instruction.

- Upon the completion of last step, the control goes back to step 1 to start over the cycle for the next instruction.
- The process is repeated indefinitely unless a HALT instruction is encountered.



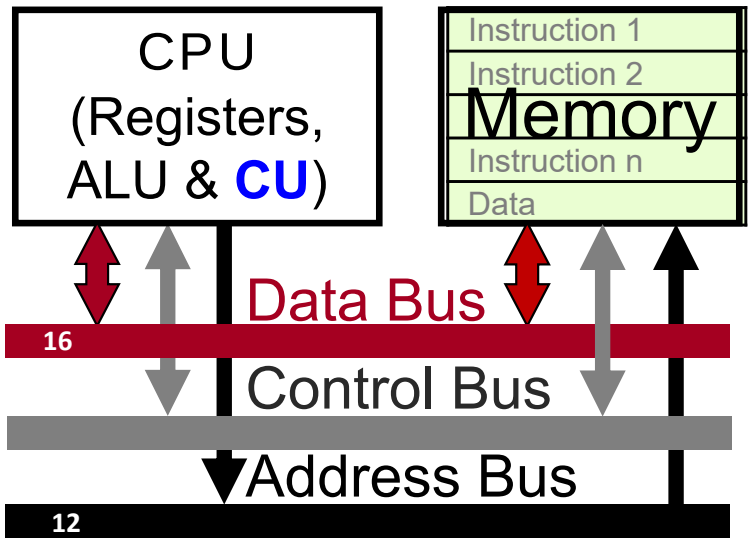
- Computer performs the instruction cycle forever! (or at least until it is turned off, faces an error or is instructed to STOP)

Detailed Instruction Cycle



Control Unit (CU) states: $T_0, T_1, T_2, \ldots$

- T_0 - Instruction address calculation
  - Determines the address of the next instruction to be executed (PC)
- T_1 - Instruction **fetch**
  - Reads the instruction from memory location into the processor - IR
- T_2 - Instruction operation decoding
  - Analyzes the instruction to determine the type of operation to be performed and the operand(s) to be used
- T_3 - Operand address calculation
  - Determines the operand address (if needed) - RA
- T_4 - Instruction **execution** Operand fetch
  - Fetches the operand from memory or reads it from I/O → AC
- T_5 – Instruction **execution** Data operation
  - Performs the operation indicated in the instruction (ALU)
- T_6 - Instruction **execution** Operand store
  - Write the results into memory or out to I/O



The control unit **(CU)** that implements the above steps has a similar structure like the example ASM#5.3 from the previous ASM section.

1. **Fetch** instruction from memory location specified by **PC** to Instruction Register (**IR**)

T_0: AR ← [PC] - Transfer contents of PC to Memory Address Register

T_1: IR ← [M(AR)] - Fetch instruction = contents of addressed memory location [M(AR)] is transferred to the IR

PC ← PC+1 - PC incremented to point to the next instruction

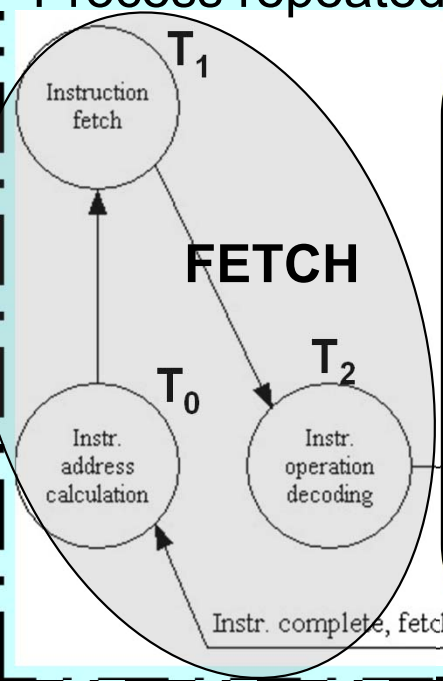
ASM#1

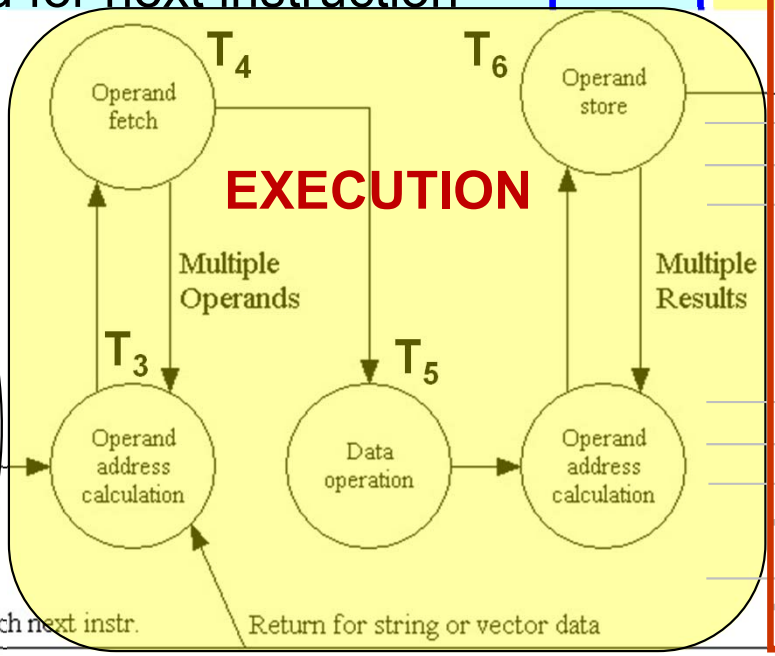
T_2 Instruction decoded by **CU**

2. **Execute** instruction held in IR

T3 …

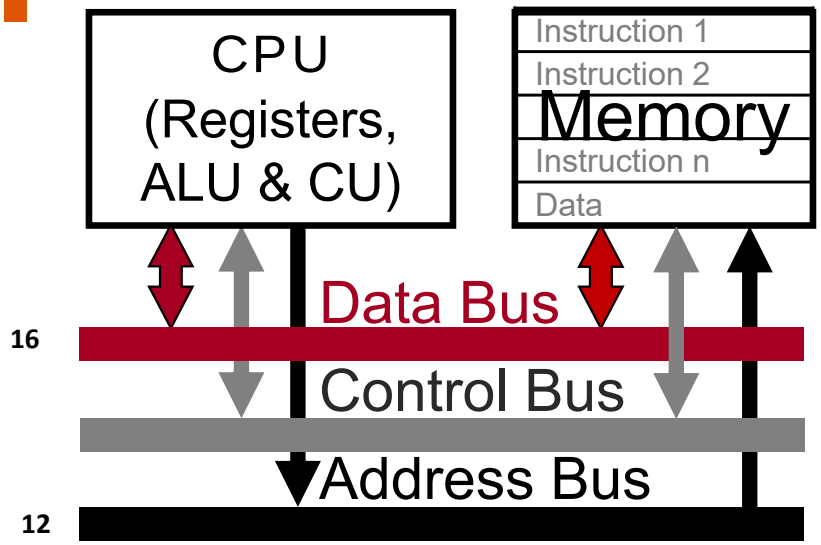
Process repeated for next instruction





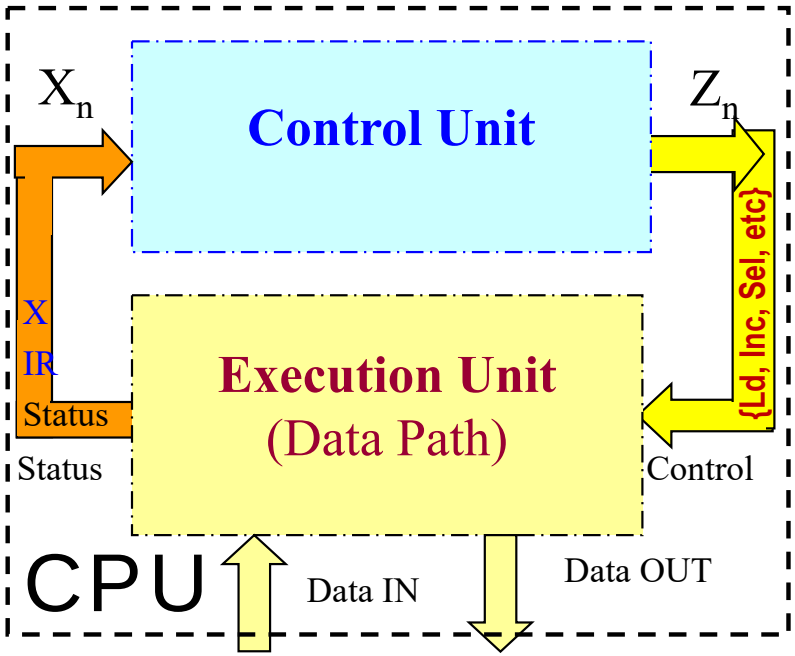
Top-Down Design

uOttawa



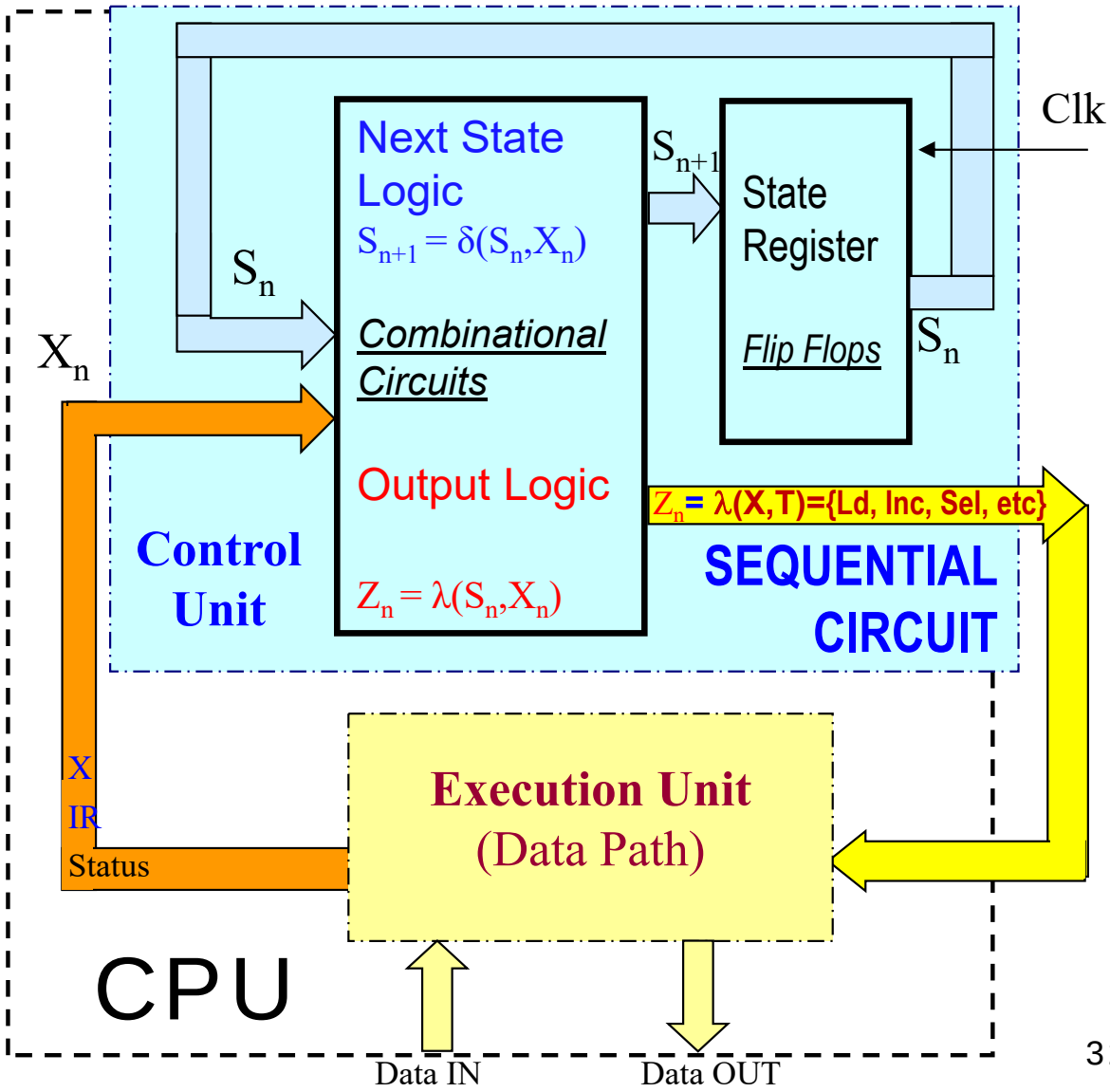
Central Processing Unit

CPU Block Diagram



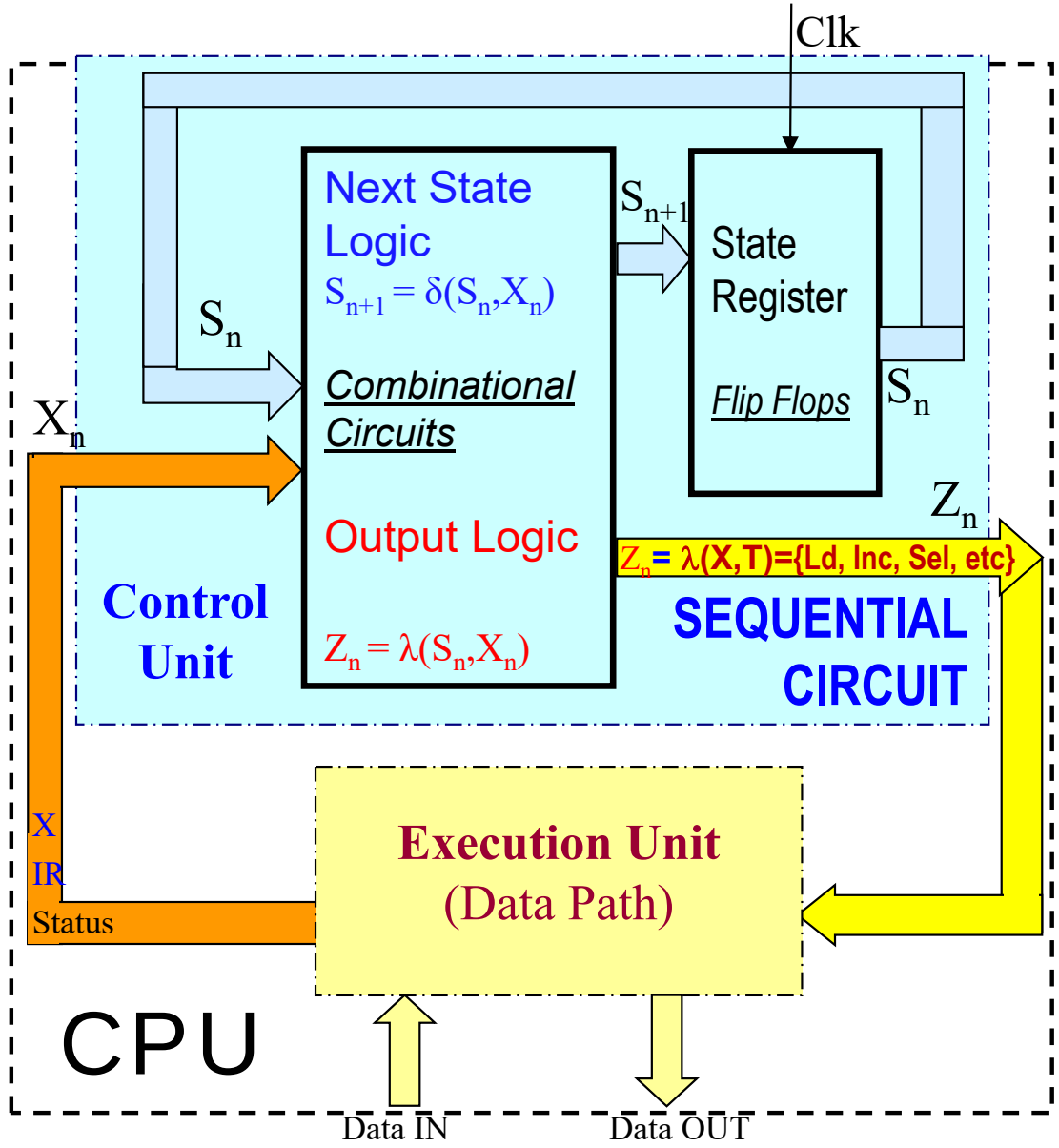
CPU **CONTROL UNIT**

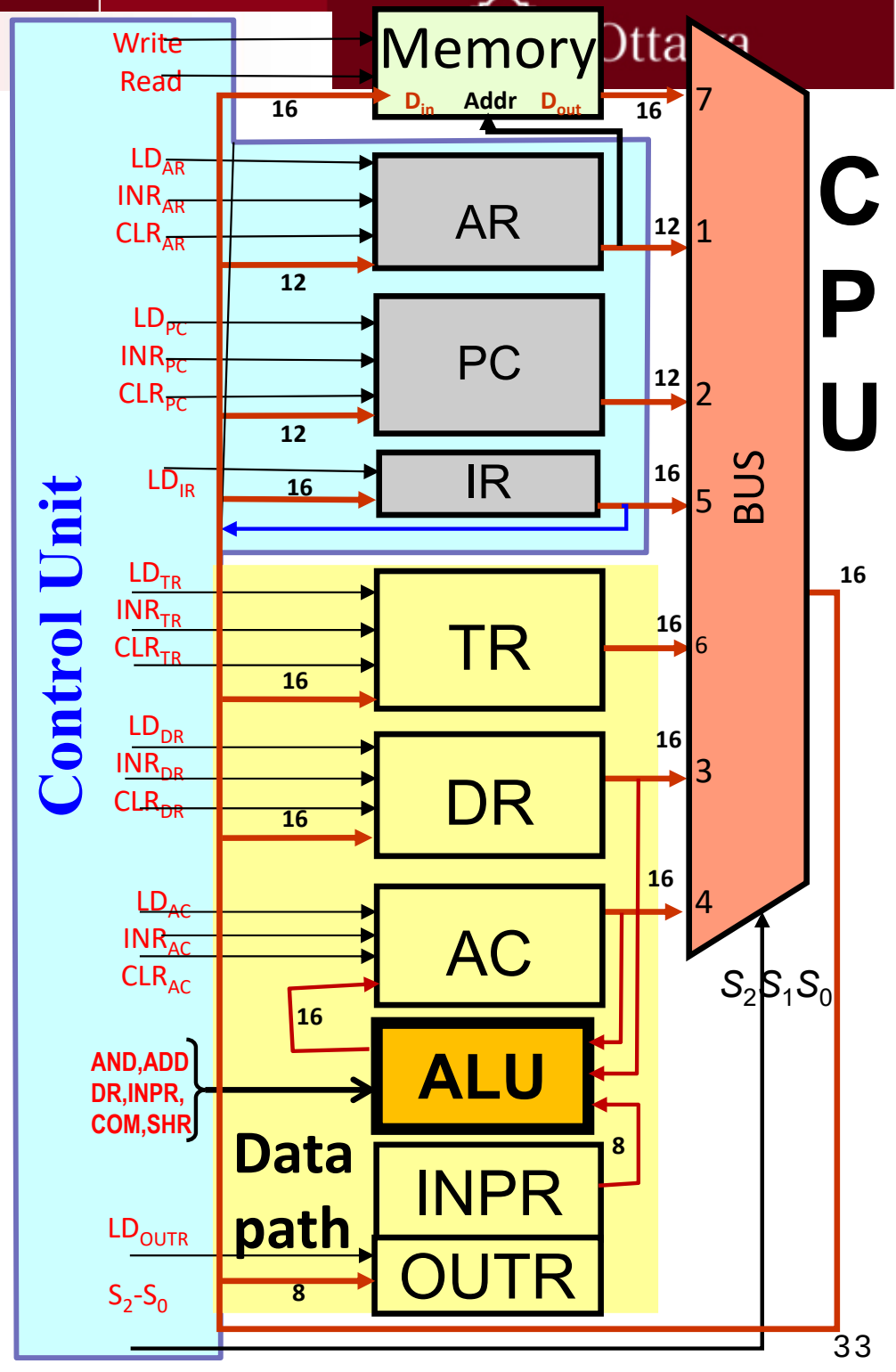
as a Sequential Circuit



Top-Down Design

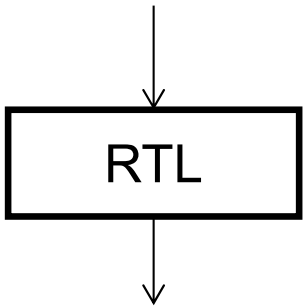
CPU CONTROL UNIT as a Sequential Circuit



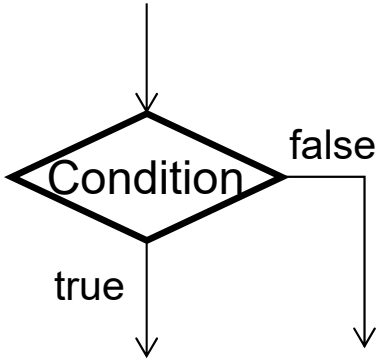


ASM#2 - High Level *RTL* ASM Flowcharts Constructs

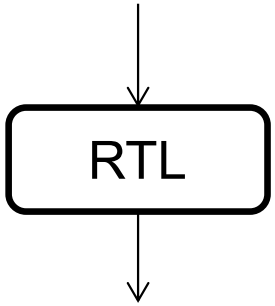
- Original ASM constructs consisted of **state** boxes, **condition** boxes and **conditional output** boxes
  - Conditional output boxes (Mealy FSM's) are difficult to represent in text or table ASM chart format, and allow conditional data transfers (RTL) … can be avoided
- ASM textual or tabular charts have replaced flowcharts because of their ease of modeling complexity and representing large synchronous sequential circuits



State box



Condition box



Conditional output box

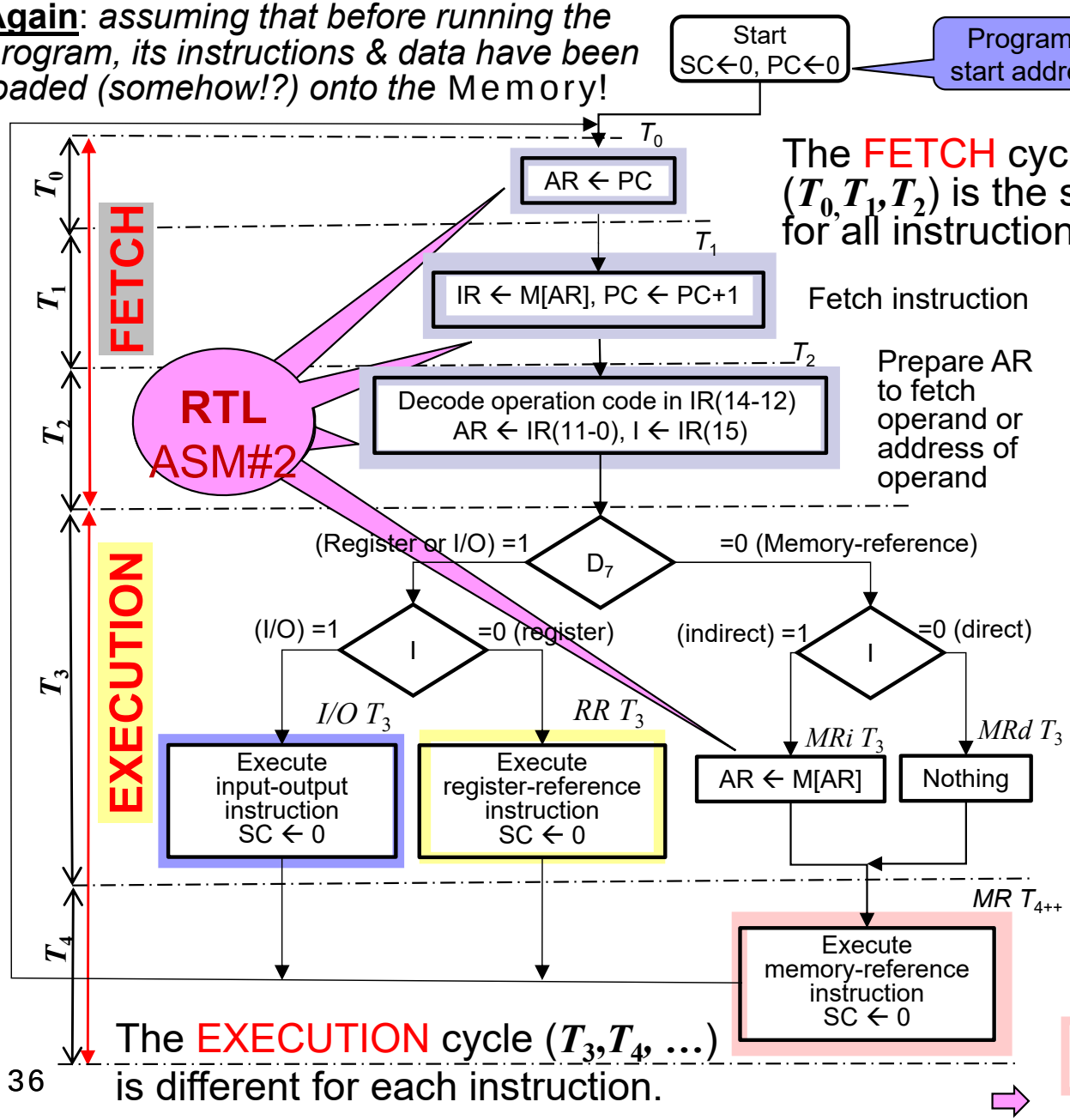
ASM Flowchart Rules (ASM#2)

- There are some rules that have to be abided by during ASM design
  1. State boxes should contain only RTL statements, control signals in parentheses, and state-transfer statements in parentheses
  2. All operations within a state box should be concurrently executable in one clock cycle
  3. If the operations in two consecutive state boxes can be executed in the same clock cycle, then the two state boxes can be combined into one state box
  4. Condition boxes should contain only simple queries that can be evaluated using purely combinational logic
  5. A new register must be assigned for each unique name in the set of RTN statements
  6. For each register-transfer statement, there must be a path between the source and the destination registers (if a transformation takes place during the transfer, then a combinational device, such as an adder or ALU, must be inserted into the path between the source and destination registers)
  7. If there are several paths leading into a combinational device or register, a multiplexer or tri-state buffer must be used
  8. For each simple binary query in a condition box, a combinational device or status signal must be used
  9. Finally, control signal inputs must be attached to each register and multiplexer so that register transfers can be precisely controlled

uOttawa

Instruction Cycle **RTL ASM#2 Flowchart**

Again: *assuming that before running the program, its instructions & data have been loaded (somehow!?) onto the* Memory!

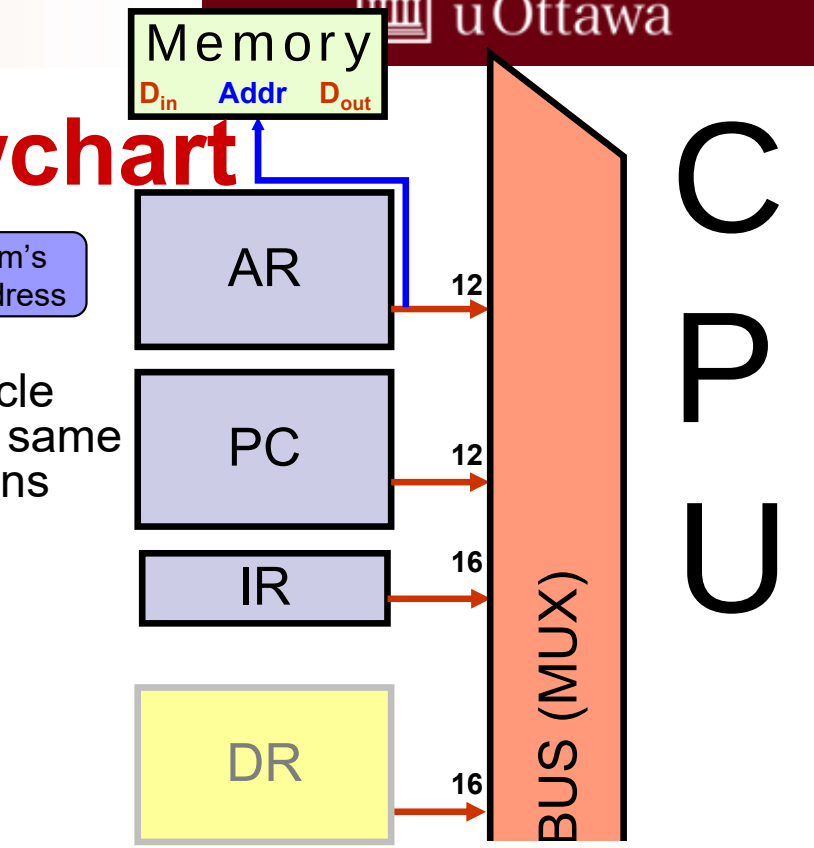


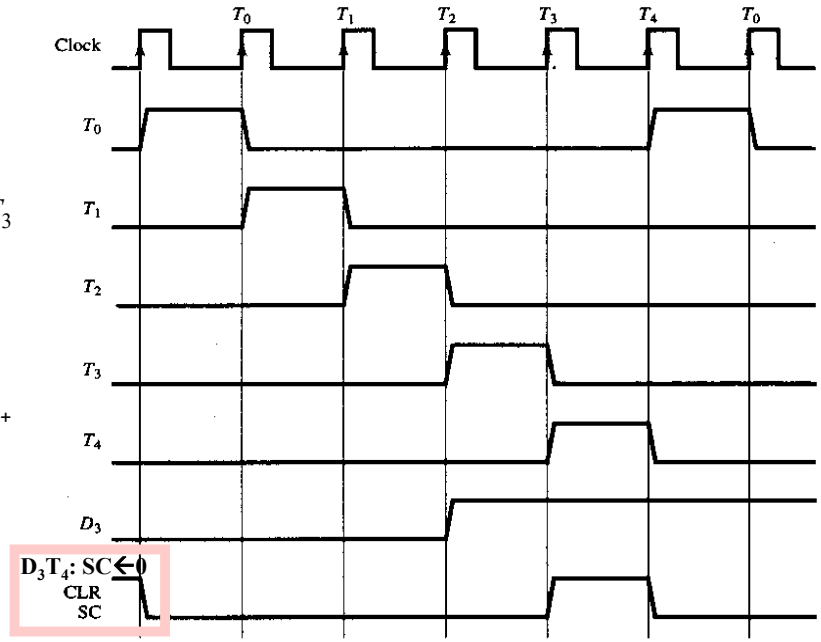
The FETCH cycle (T_0,T_1,T_2) is the same for all instructions

Fetch instruction

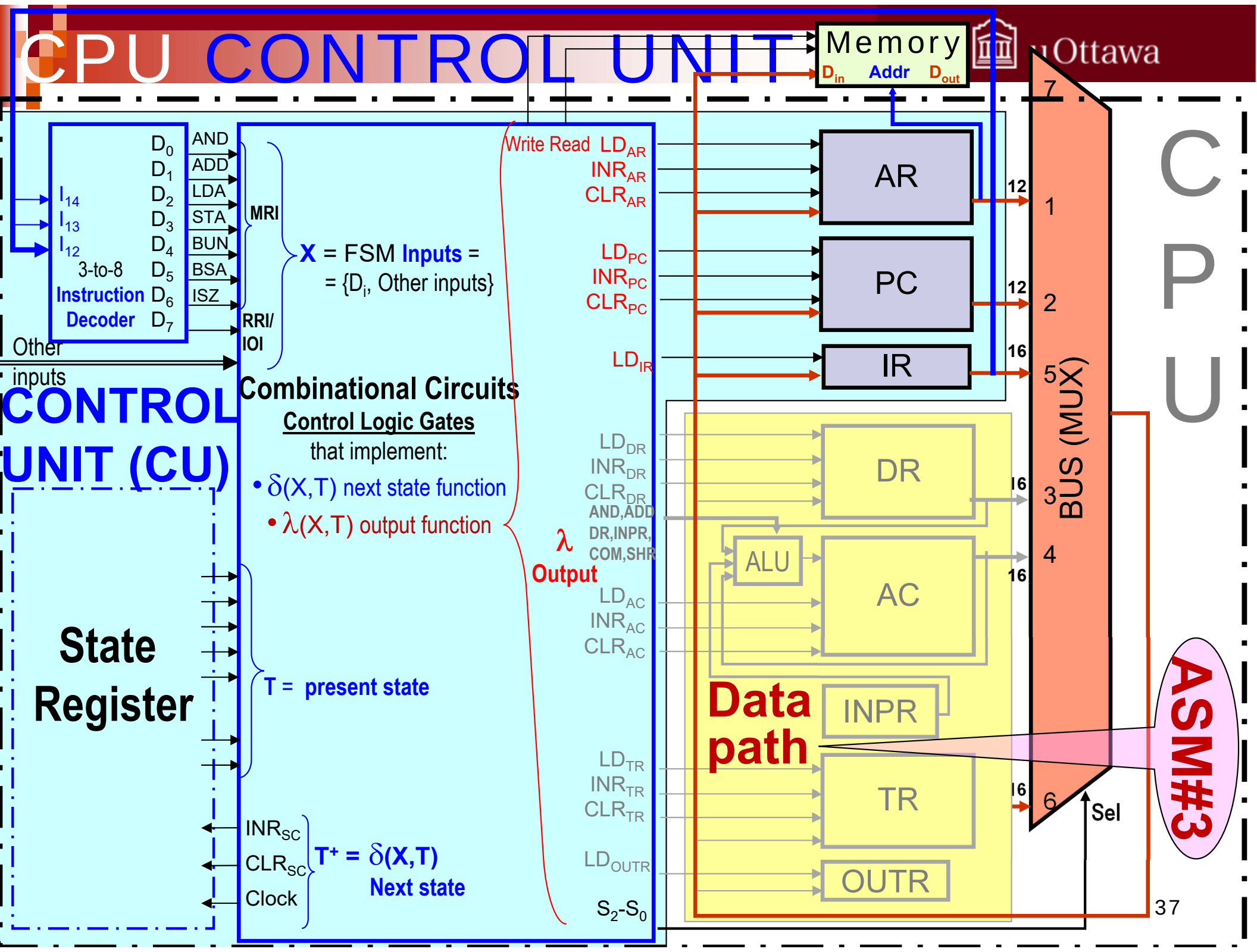
Prepare AR to fetch operand or address of operand

The EXECUTION cycle (T_3,T_4, …) is different for each instruction.

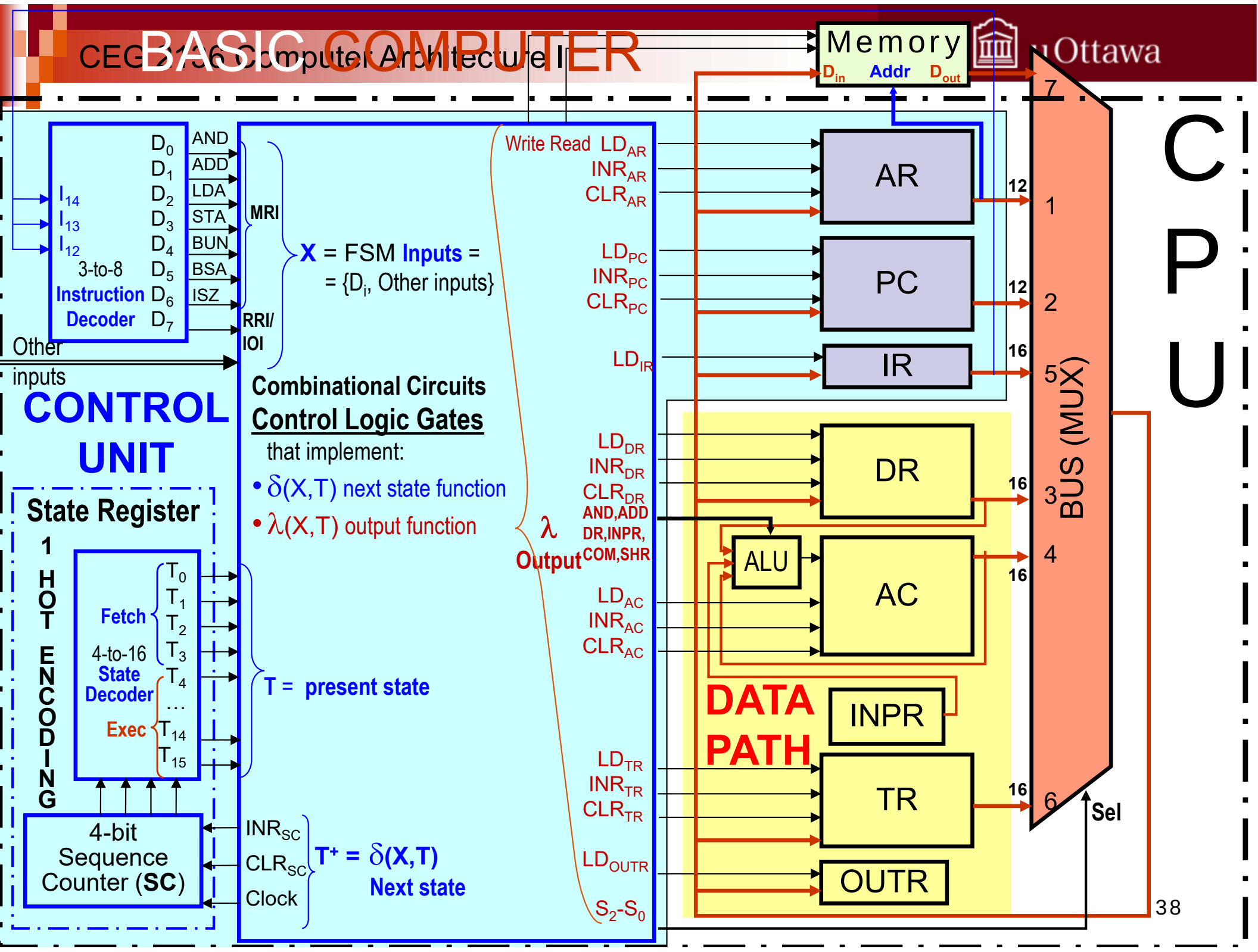




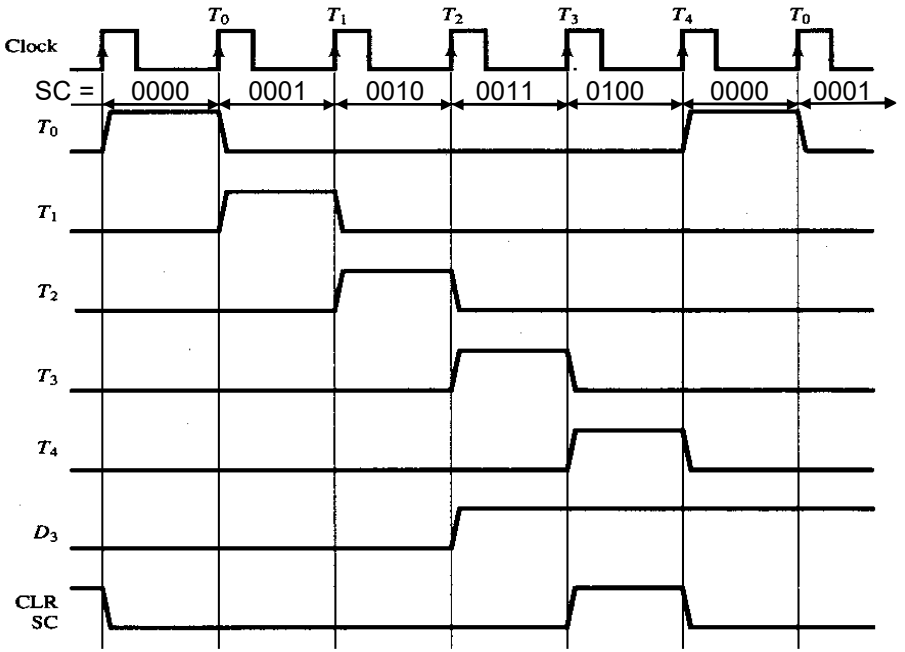
CPU CONTROL UNIT

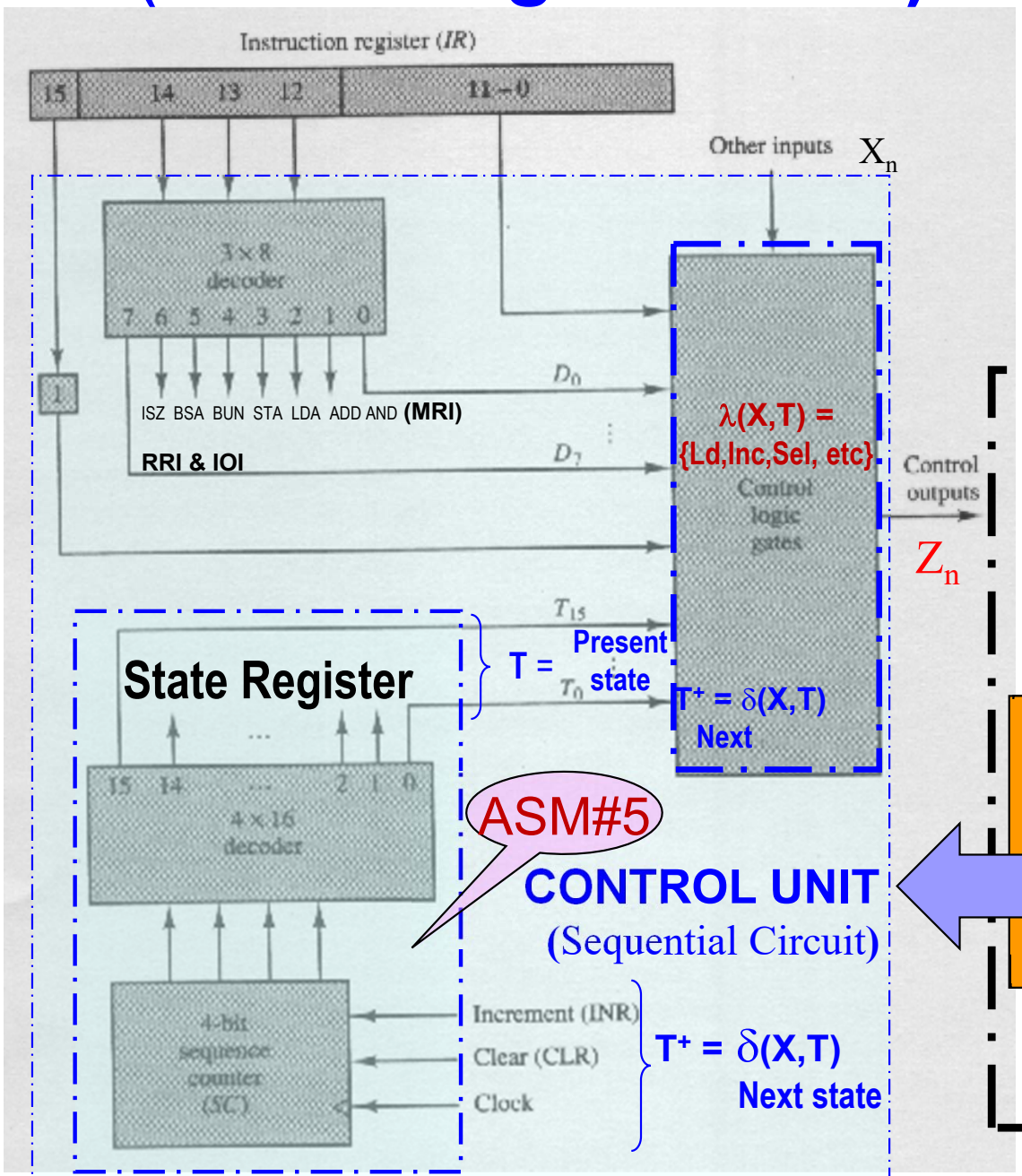


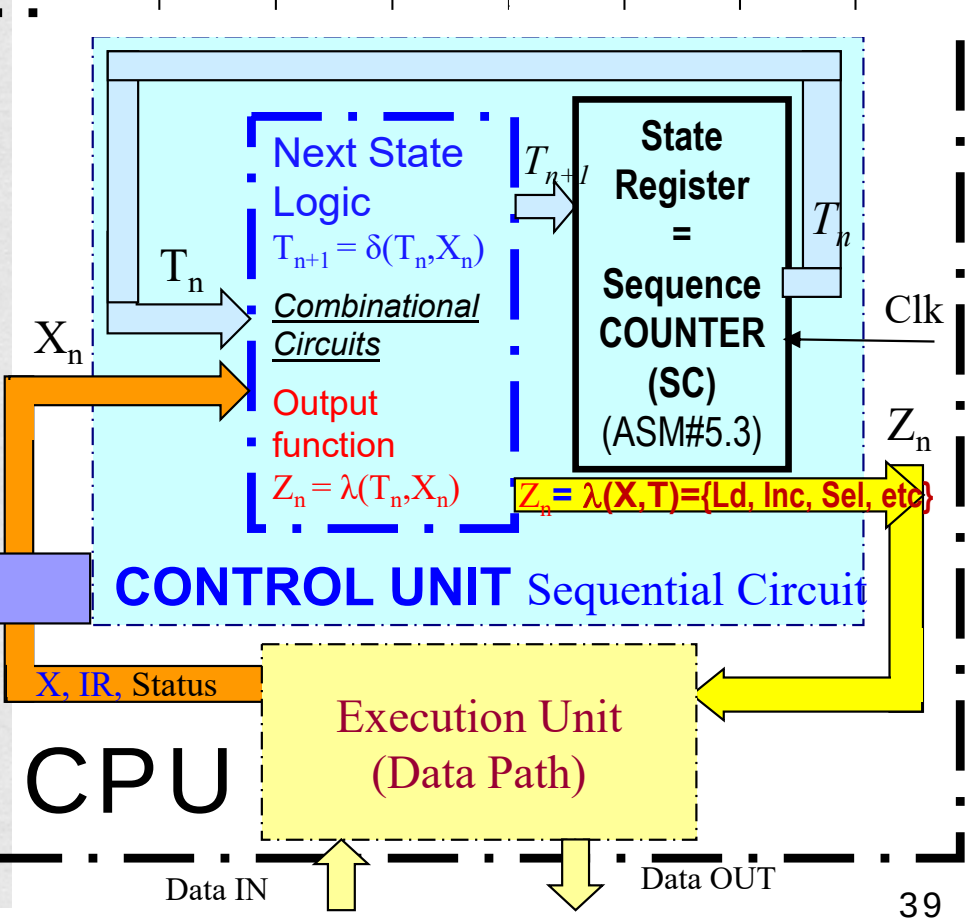
BASIC COMPUTER



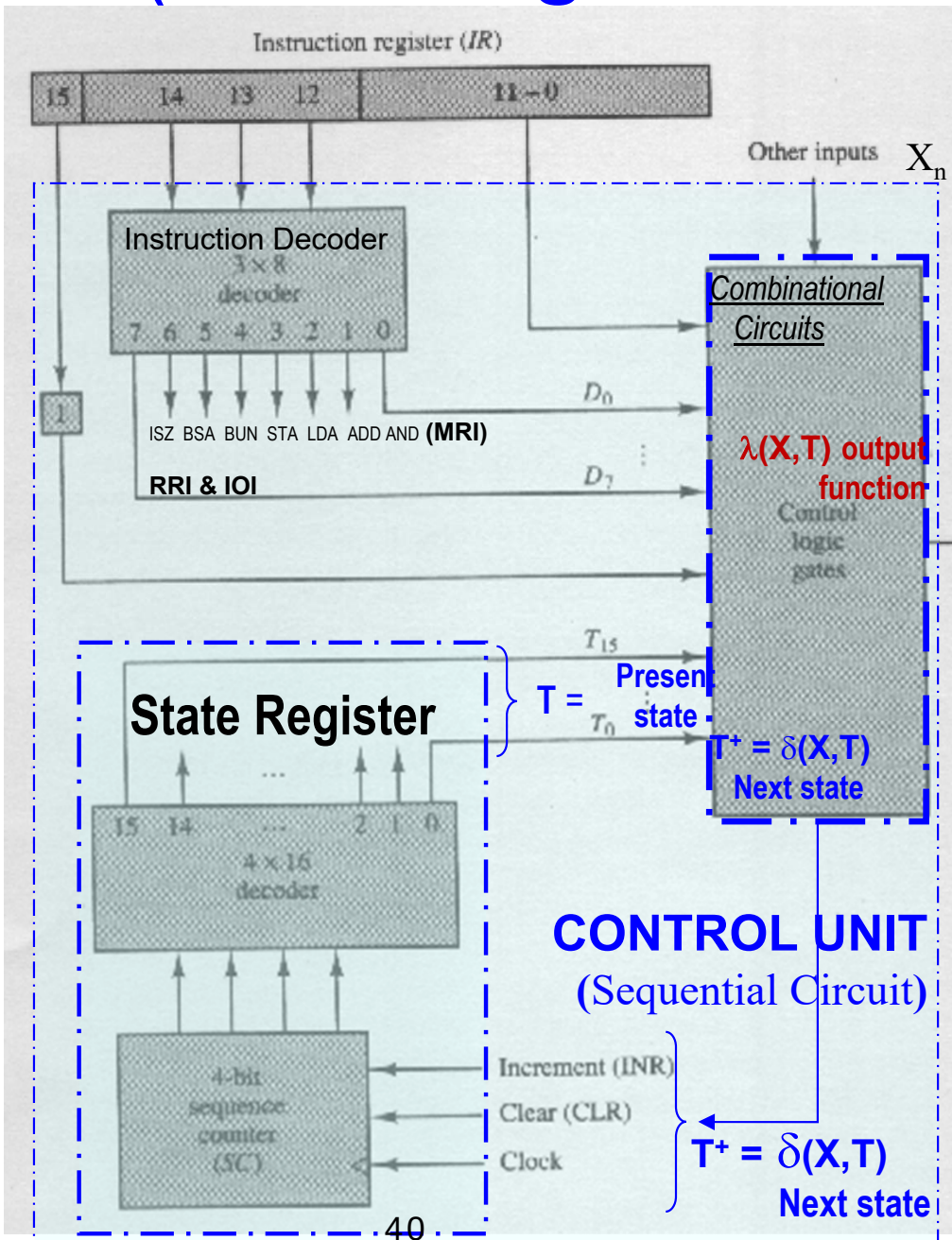
CPU CONTROL UNIT
(State Register + δ)







CPU CONTROL UNIT (State Register + δ)

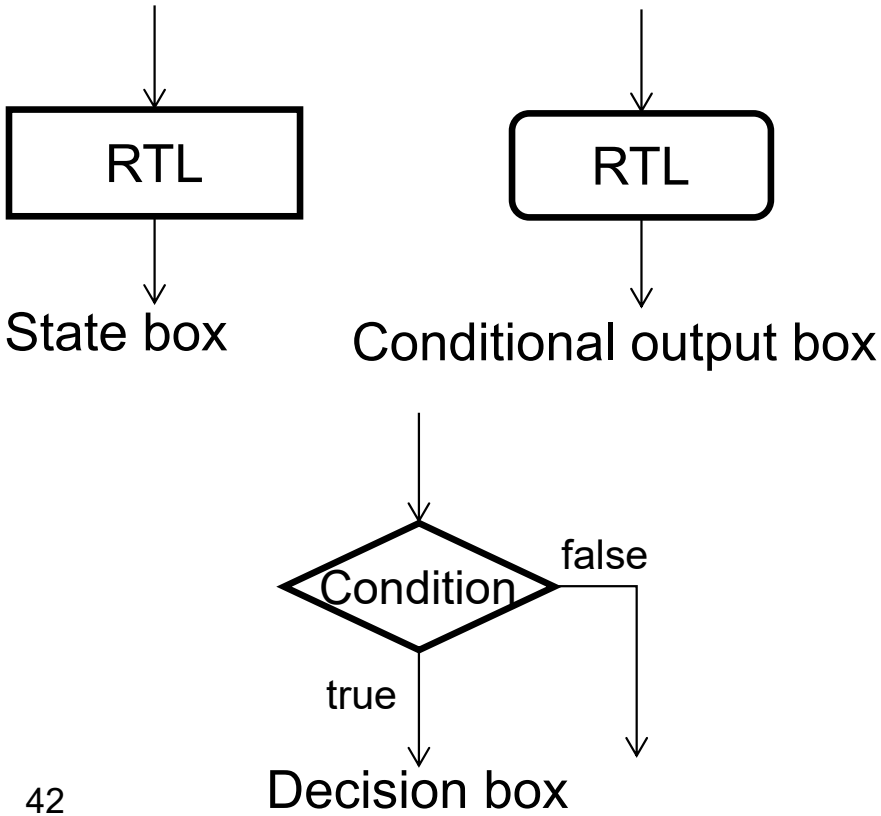


- The control unit (CU) is a *programmable sequential circuit*, whose transition (δ) and output (λ) functions changes in accord with the current instruction I_{14}-I_0.
- CU consists of a State Register (sequence counter SC + 4x16 decoder) & Combinational Circuits that compute the *output function* (λ - control signals for the multiplexers and registers in the system) & the *transition function* (δ - CLR clears SC at the end of the execution of every instruction to return SC to the initial state T_0, for the next instruction).
- The current instruction is read from memory and it is stored in the instruction register *IR* during its execution.
- The 3-bit opcode $IR_{14}IR_{13}IR_{12}$ is decoded using a 3 x 8 decoder (*Instruction Decoder*);
- the outputs of the *Instruction Decoder* are D_0 - D_7, each corresponding to one operation.
- IR_{15} is transferred to a flip-flop *I*.
- *SC* counts in binary from 0-15. The outputs of *SC* are decoded into 16 timing signals T_0-T_{15}
- The 12 least significant bits of *IR*, *I*, D_0-D_7, and T_0 - T_{15}, are all passed as inputs to the Control Logic Gates, which implements **δ(X,T)**=CLR SC and **λ(X,T)=**{Ld, Inc, Sel, etc}

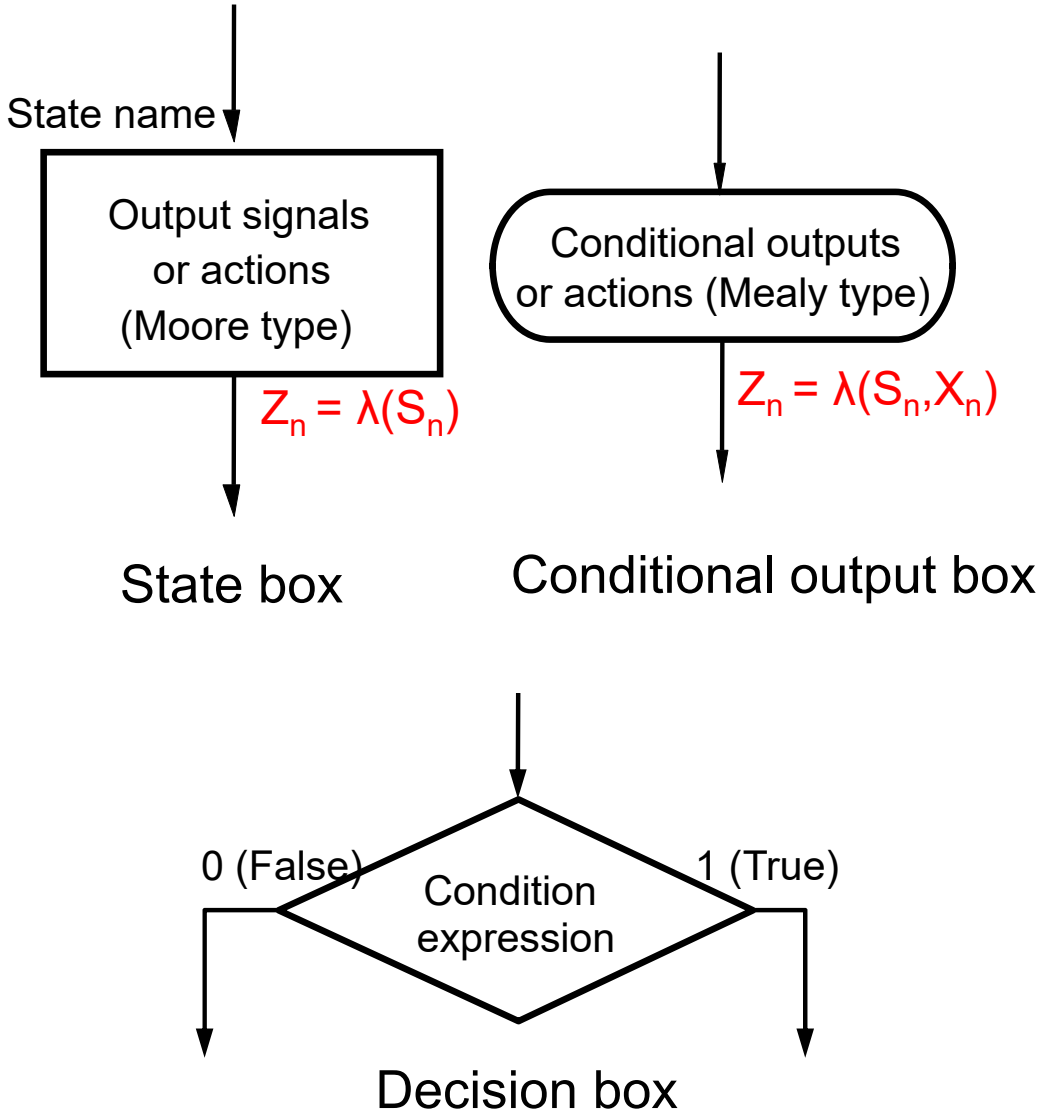
Detailed specifications of instruction FETCH in RTL ASM (ASM#2) and with detailed ASM (ASM#4)

ASM#2 **Flowcharts**

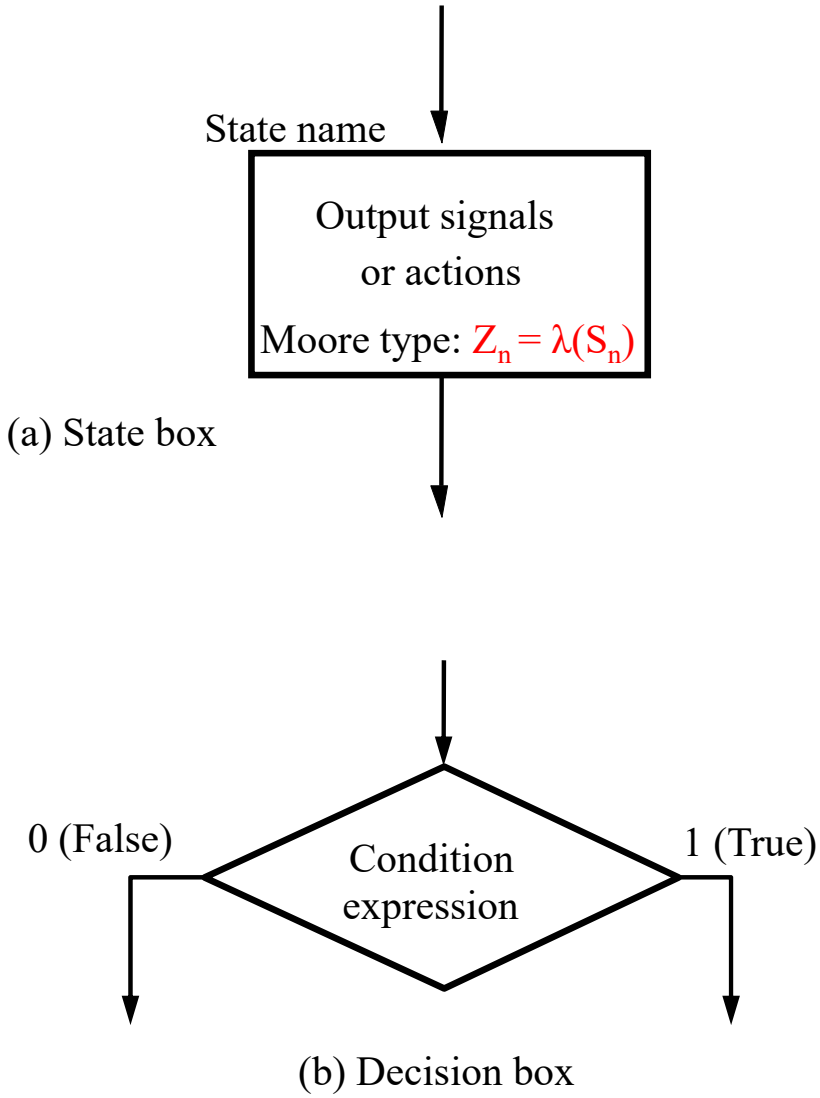
ASM textual or tabular charts replace flowcharts because of their ease of modeling complexity and representing large synchronous sequential circuits.

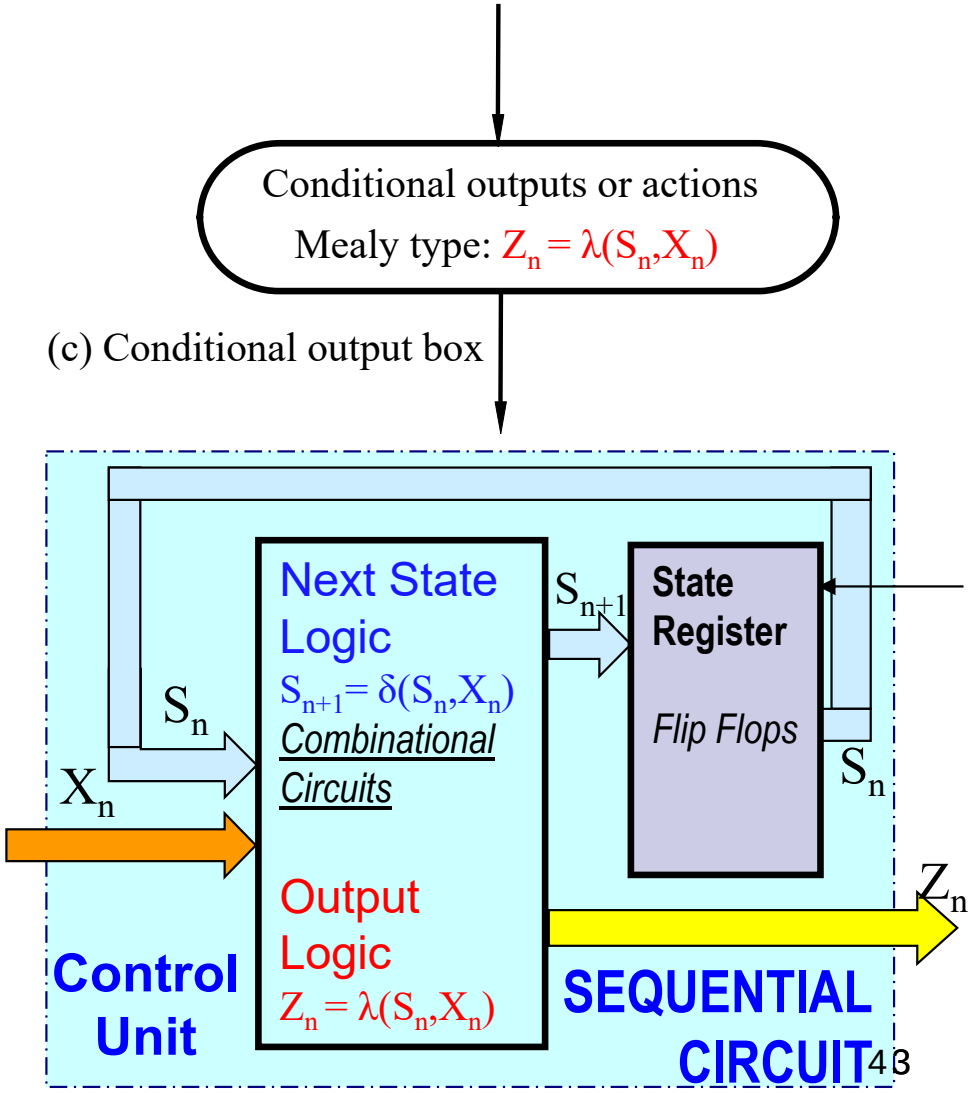


ASM#4 **Detailed Chart**



Symbols used in **ASM#4 detailed charts**



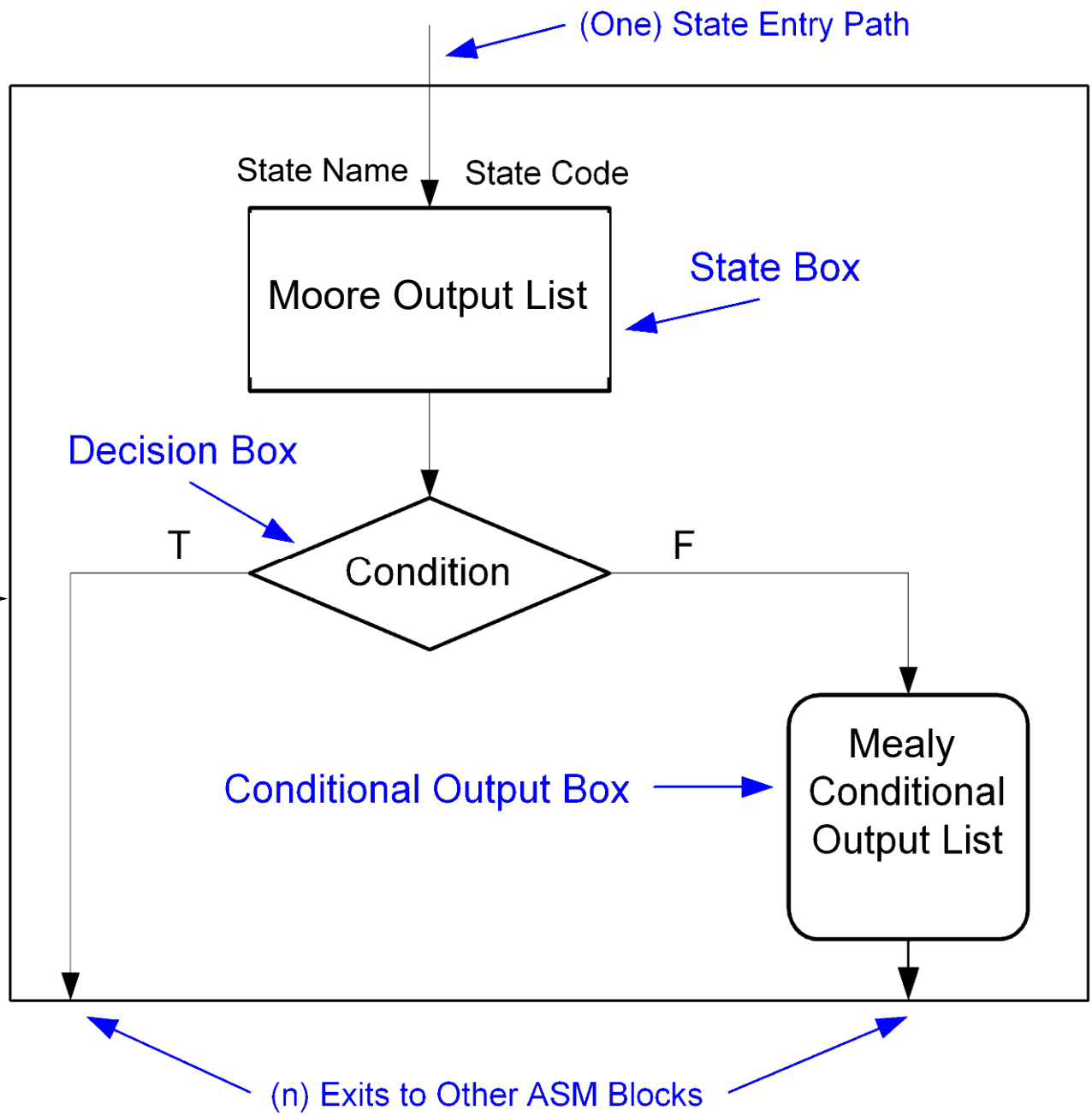


ASM Basic Block

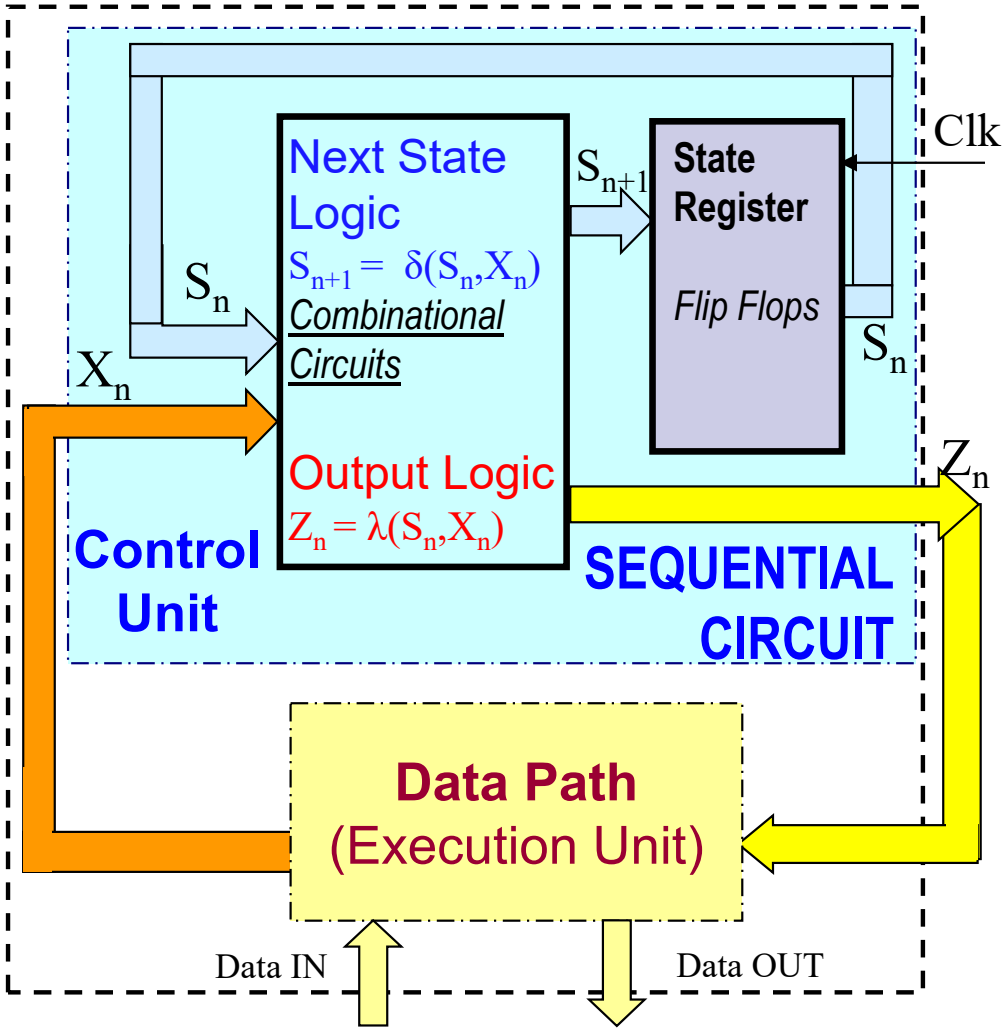
Recall the main steps in the ASM methodology:

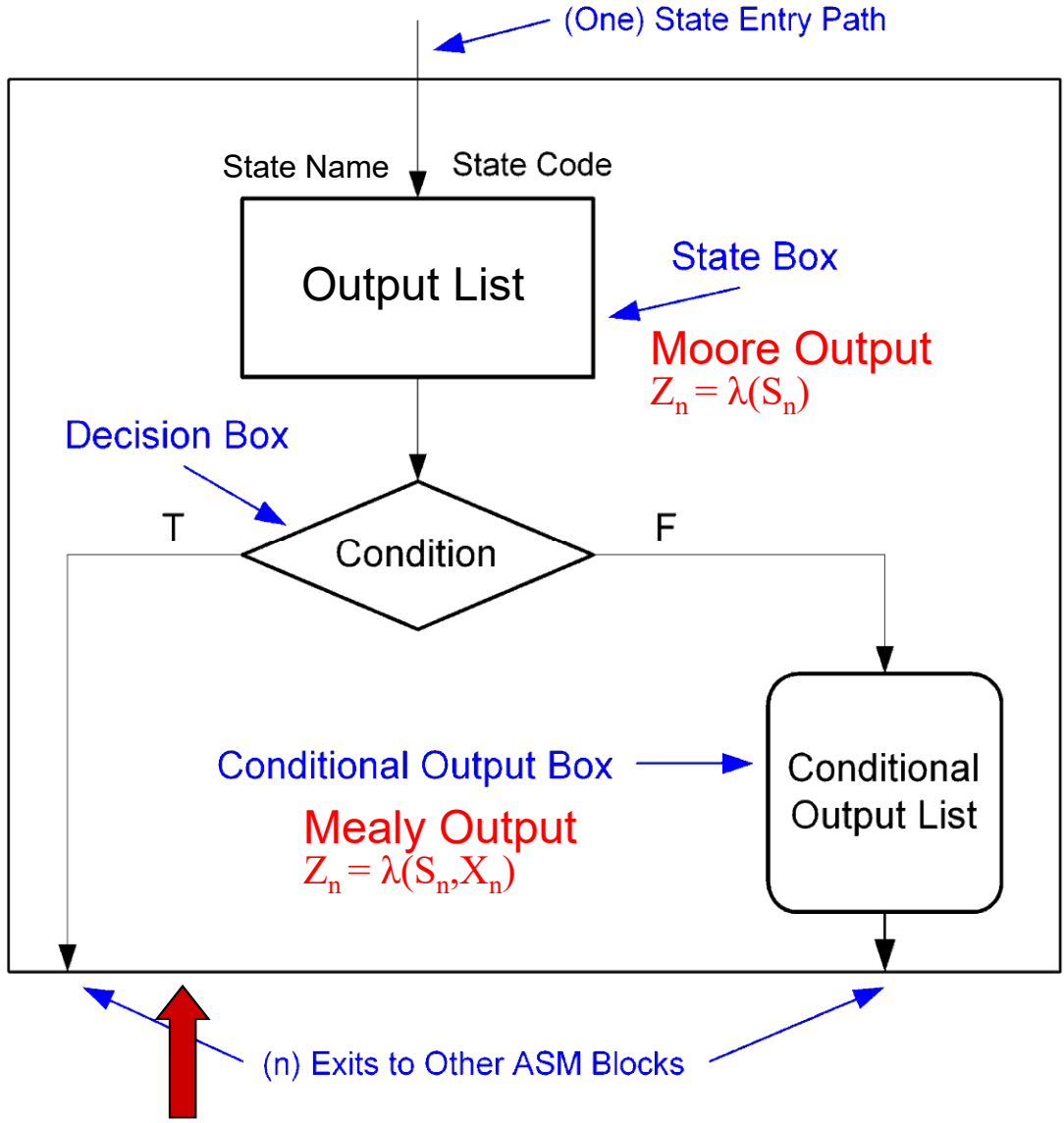
- Create an algorithm, using *pseudocode*, to describe the desired operation of the device
- Convert the pseudocode into an *ASM chart* (the basic brick = **ASM block**)
- Design the *datapath* based on the ASM chart
- Create a *detailed ASM chart* based on the datapath
- Design the *control logic* based on the detailed ASM chart

Combination of **datapath** and **control logic** makes up the actual logic system that will solve the original problem.



ASM#4 Detailed Chart





The basic brick = **ASM block** Combination of **datapath** and **control unit** makes up the actual logic system.

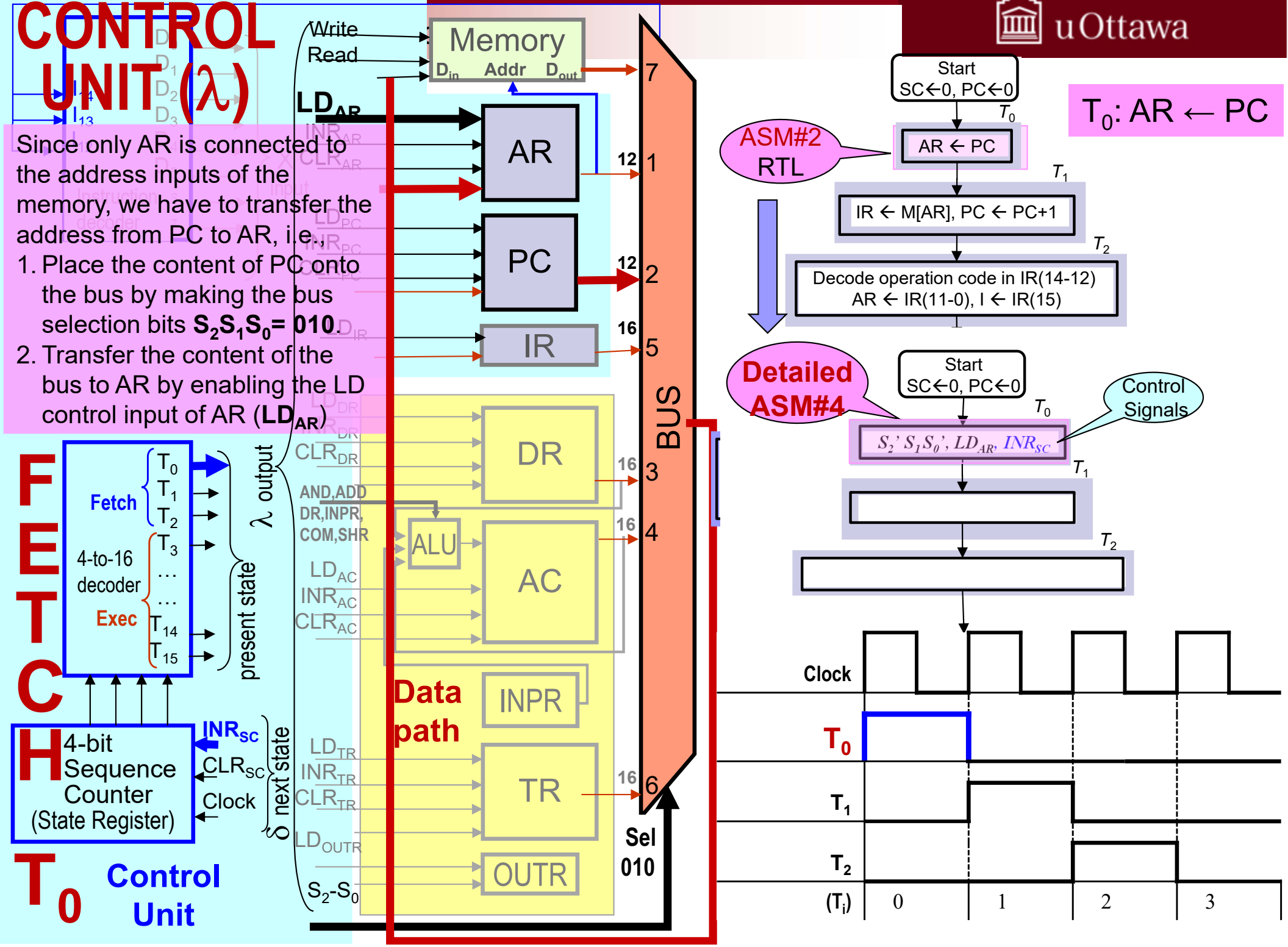
ASM#4 Detailed Chart

- There are some rules that have to be abided by during detailed ASM chart design
  1. For each *box* in the ASM flowchart, there should be a corresponding *state box* in the *detailed ASM chart*
  2. For each RTL operation in a state box of an ASM chart, the corresponding state box of the detailed ASM chart should contain an *output list* of control signals that are applied to the datapath in order to accomplish the RTL micro-operation described in the ASM chart. These control signals are generated by the *control unit* (CU) through its output function $Z = \lambda(S,X)$ - as CU is a sequential circuit.
     - All control signals are assumed to have the value '1' if specified and the value '0' if left unspecified.
  3. For each condition box in the ASM chart, there should be a corresponding condition box in the detailed ASM chart
     - Each condition box in the detailed ASM chart must contain a logical combination of *status signals* (coming from the datapath) that implements the combinational logic query in the corresponding ASM chart condition box
- The above rules provide the mapping between the ASM chart and the detailed ASM chart
- Pay special attention to the state box and conditional box mappings!
- Control logic is the heart of the digital circuit, and is often the most complex and detailed block of the entire system, and thus must be designed **very carefully!**
- The detailed ASM chart should easily be converted to a control path, as long as the aforementioned rules are followed

CONTROL UNIT (λ)

Since only AR is connected to the address inputs of the memory, we have to transfer the address from PC to AR, i.e.,

1. Place the content of PC onto the bus by making the bus selection bits $S_2S_1S_0$= 010.
2. Transfer the content of the bus to AR by enabling the LD control input of AR (LD_{AR})

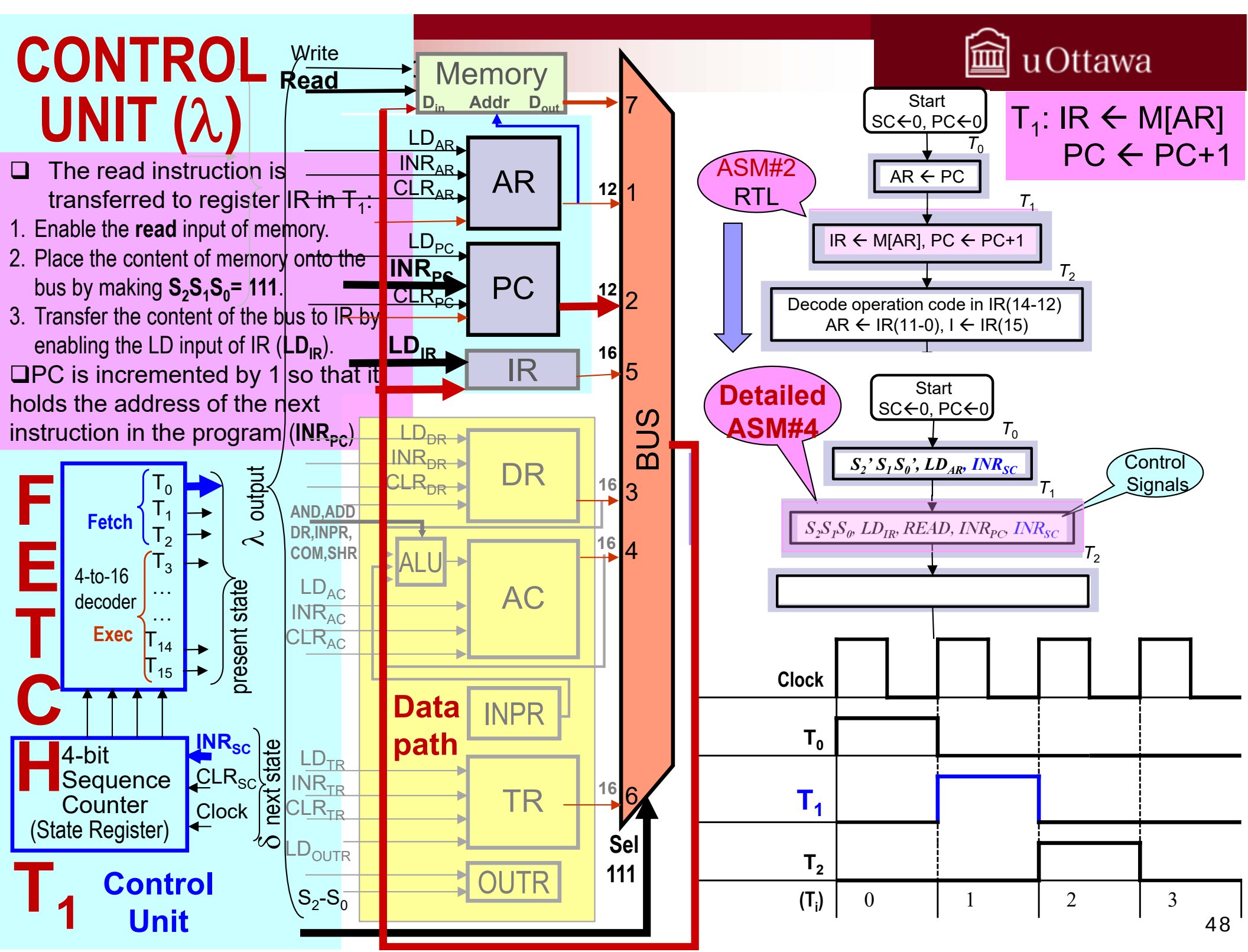


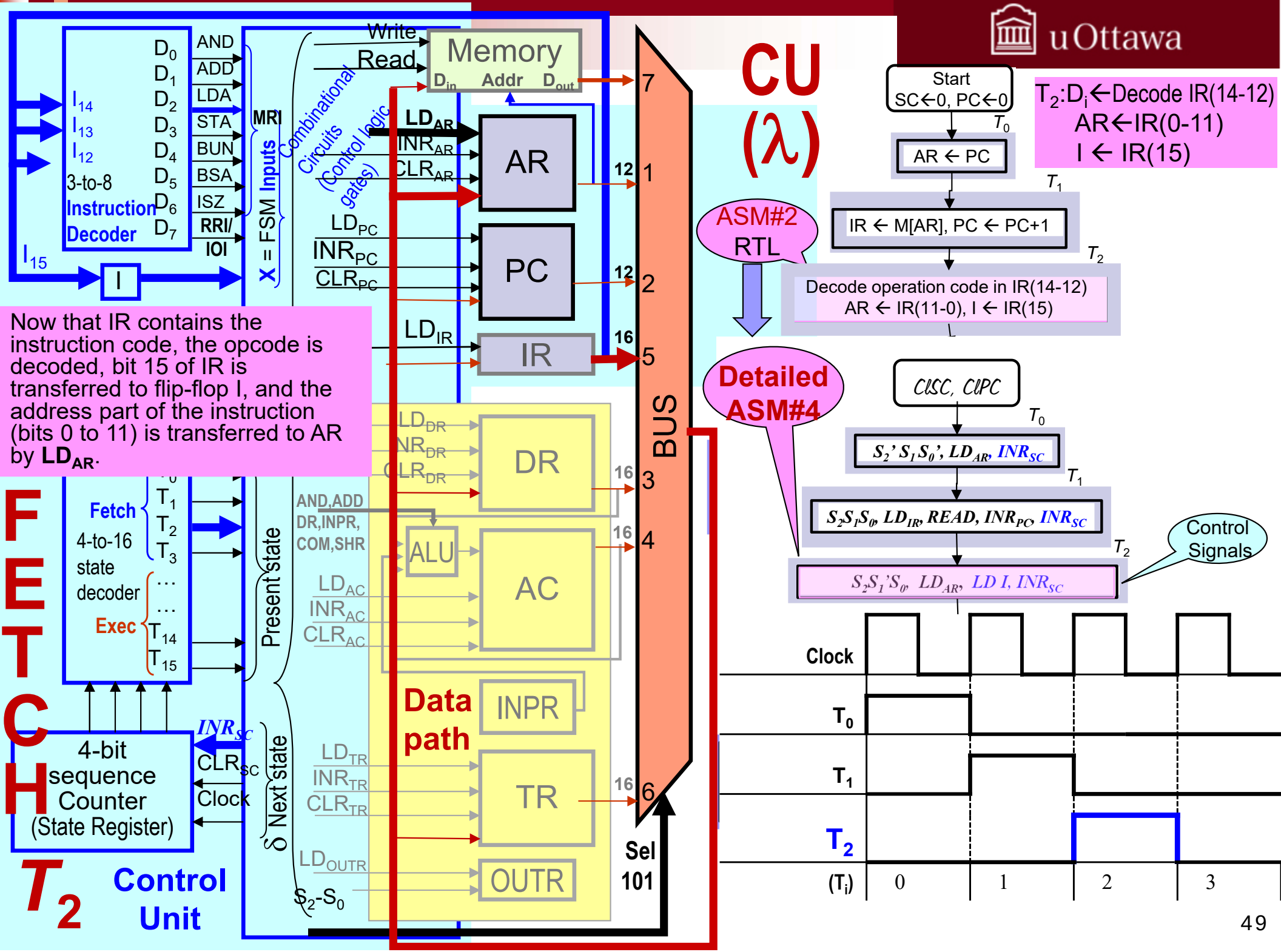
CONTROL UNIT (λ)

- The read instruction is transferred to register IR in T_1:

1. Enable the **read** input of memory.
2. Place the content of memory onto the bus by making $S_2S_1S_0$= 111.
3. Transfer the content of the bus to IR by enabling the LD input of IR (LD_{IR}).

- PC is incremented by 1 so that it holds the address of the next instruction in the program (INR_{PC})



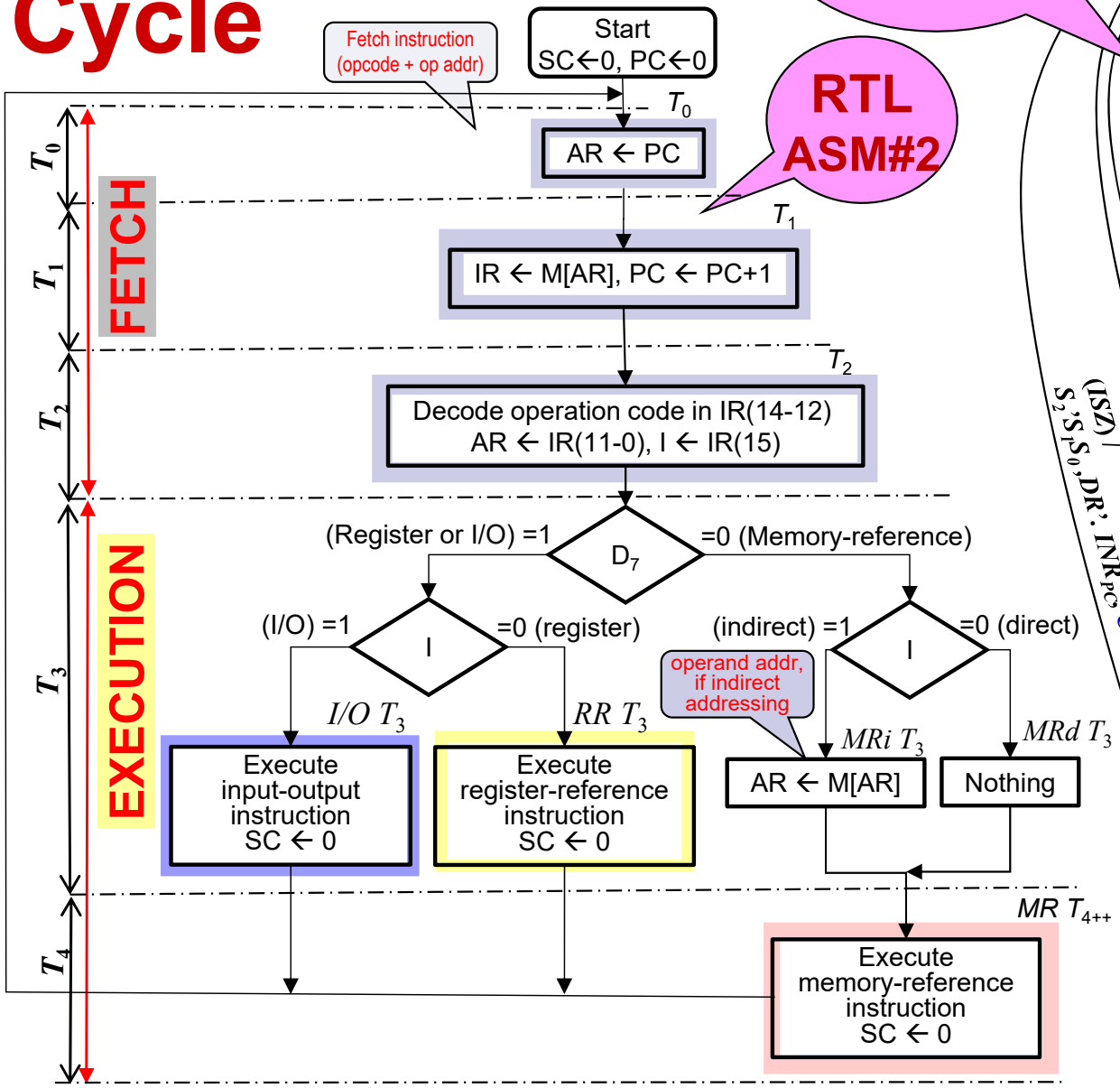


Now that IR contains the instruction code, the opcode is decoded, bit 15 of IR is transferred to flip-flop I, and the address part of the instruction (bits 0 to 11) is transferred to AR by **LD_{AR}**.

FETCH

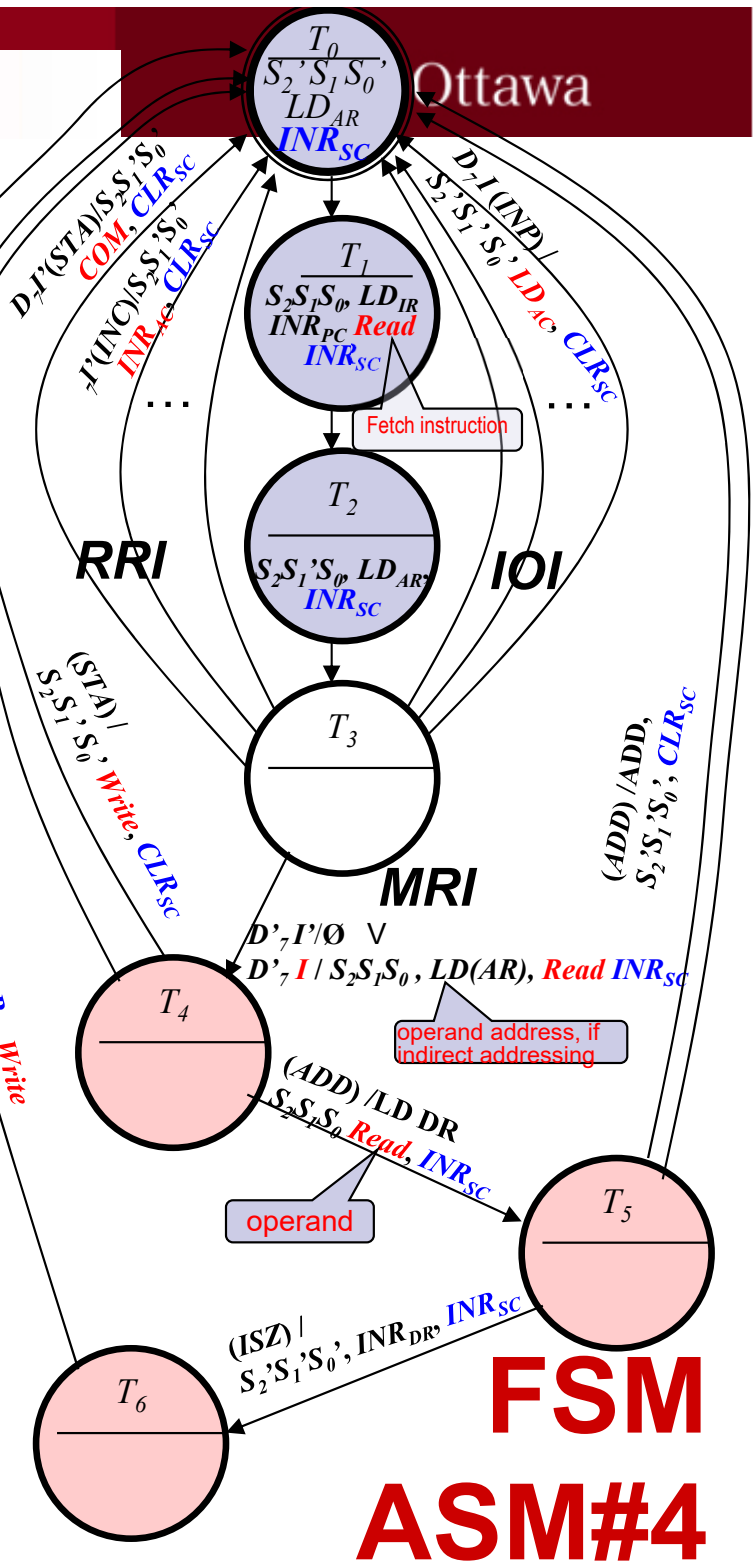
T_2 Control Unit

Instruction Cycle



(Incomplete) **Mealy FSM equivalent of ASM#4**

RTL ASM#2



FSM ASM#4

Control Unit

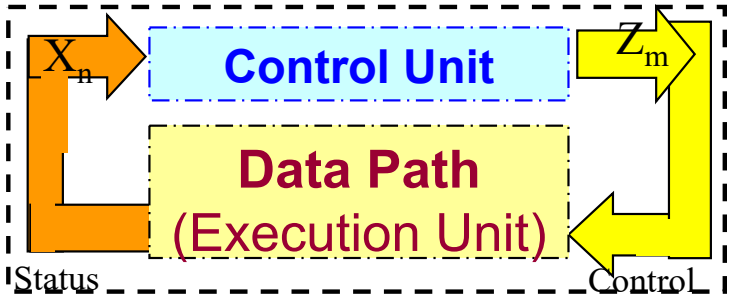
ASM#5 Synthesis

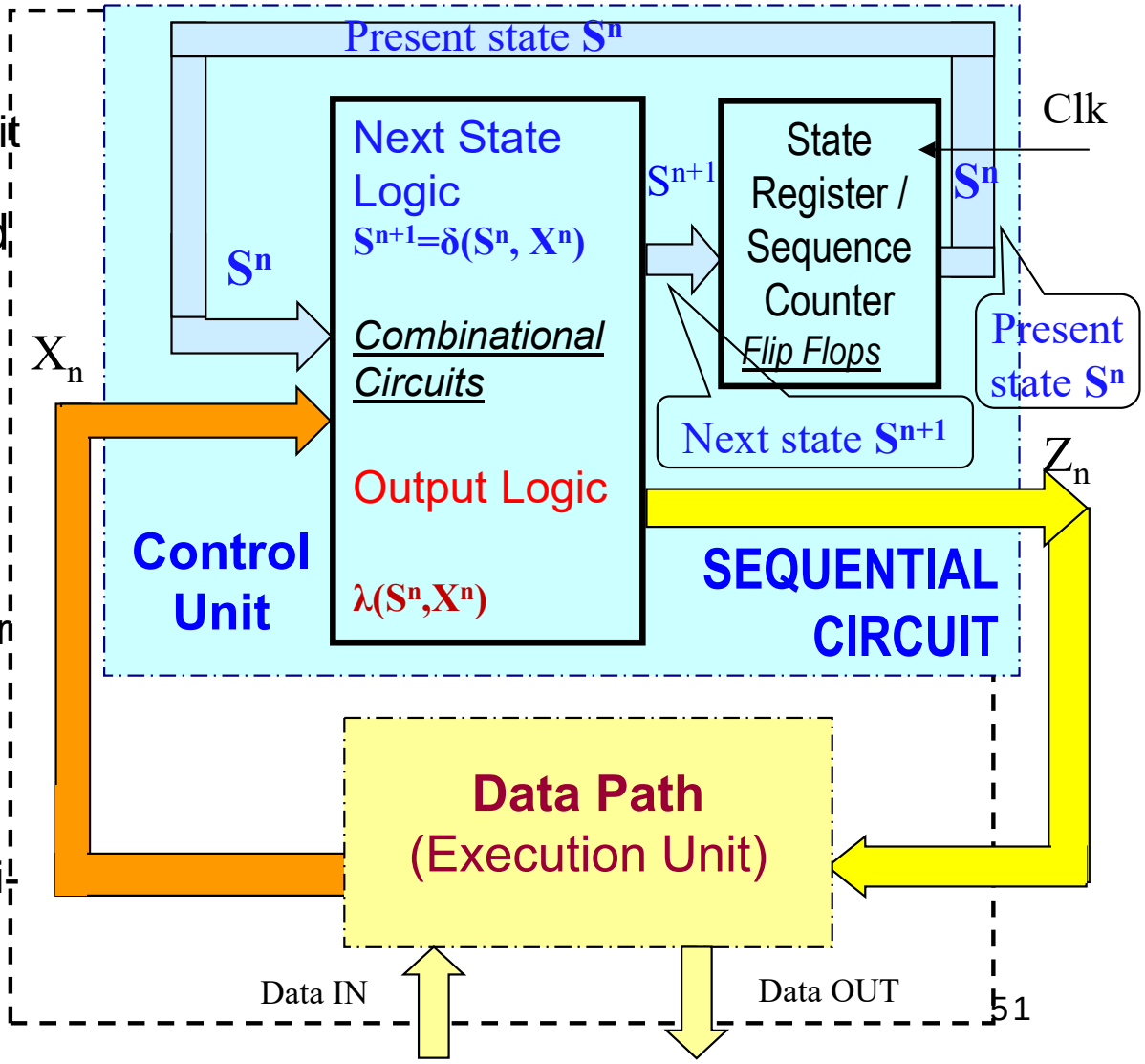
There are three major methods for implementing the State Register of sequential circuits which are used as **control logic** ASM-based circuits:

1. The **one-FF-per-state** method
   - Also called One-hot,
   - Most popular design technique, as it is simple and easy to model
2. The **Binary Encoded State** method
   - Uses state encoding to reduce the number of D flip-flops required in the state register
   - Control logic is spread out over several different digital devices
3. The **sequence-counter** method
   - Matches the cyclical nature of instruction execution on a computer
   - Control is simplified because of the cyclical state transition pattern

Control Units can be implemented as

- **Hardwired CU** - with gates and other MSI components (MUX, Decoders, multi-function registers) – Chapter 5
- **Microprogram CU** - ROM based.- Ch 7





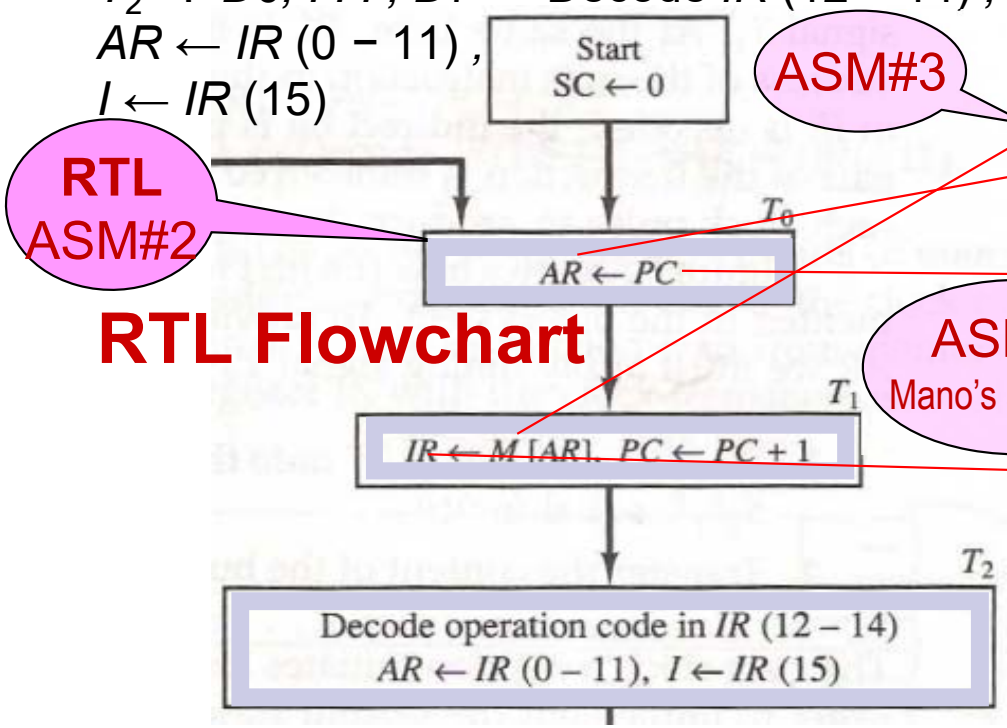
ASM#5: CU Synthesis Directly from RTL ASM#2

- When a program is executed, the *Program Counter PC* gets initially loaded with the address of the first instruction of the program, and *State Counter SC* is cleared making T_0=1
- After each clock pulse, *SC* is automatically incremented, so that the timing signals go through T_0, T_1, T_2, … and ASM states advance
- The micro-operations of the **fetch** and decode phases can be specified by the following RTL statements:

T_0 : $AR \leftarrow PC$

T_1 : $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$

T_2 : $D0, \ldots, D7 \leftarrow$ Decode $IR(12-14)$, $AR \leftarrow IR(0-11)$, $I \leftarrow IR(15)$

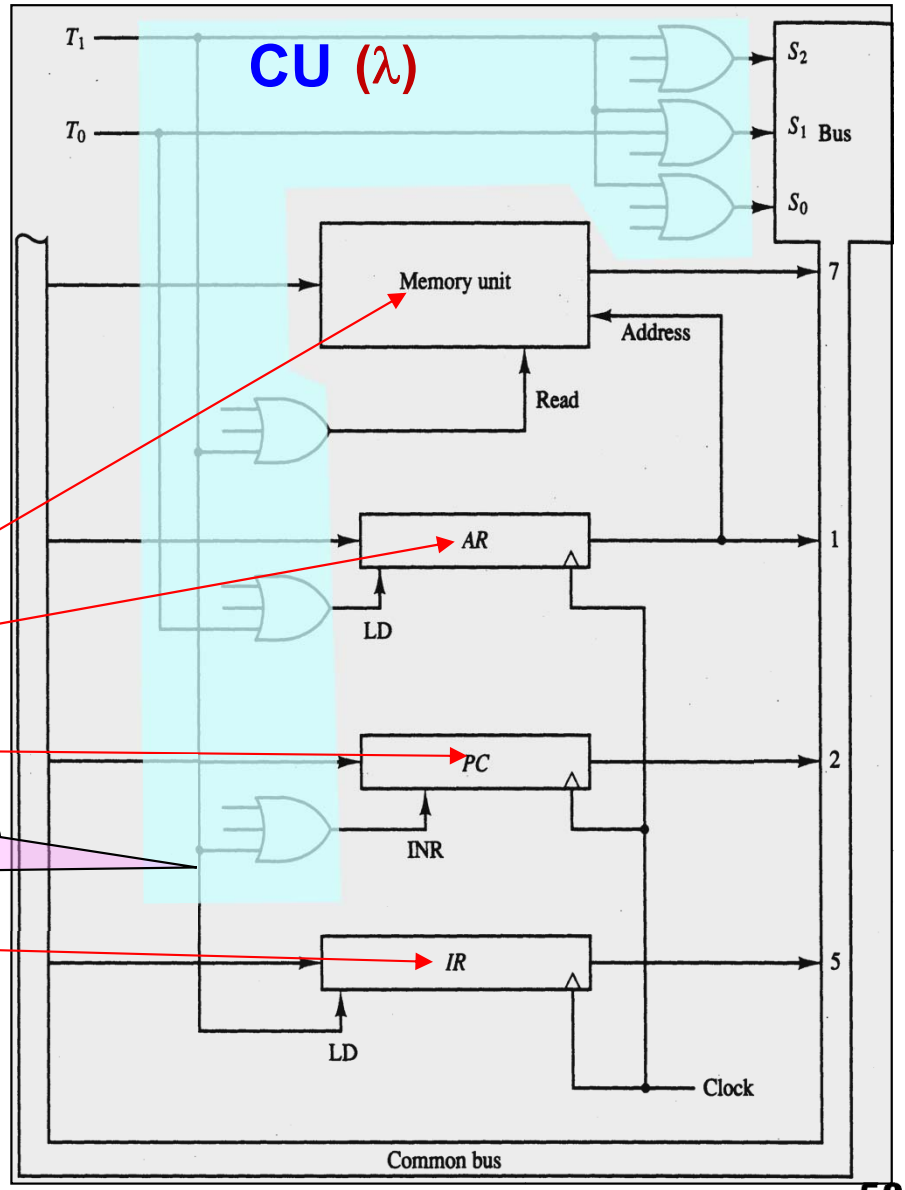


RTL ASM#2

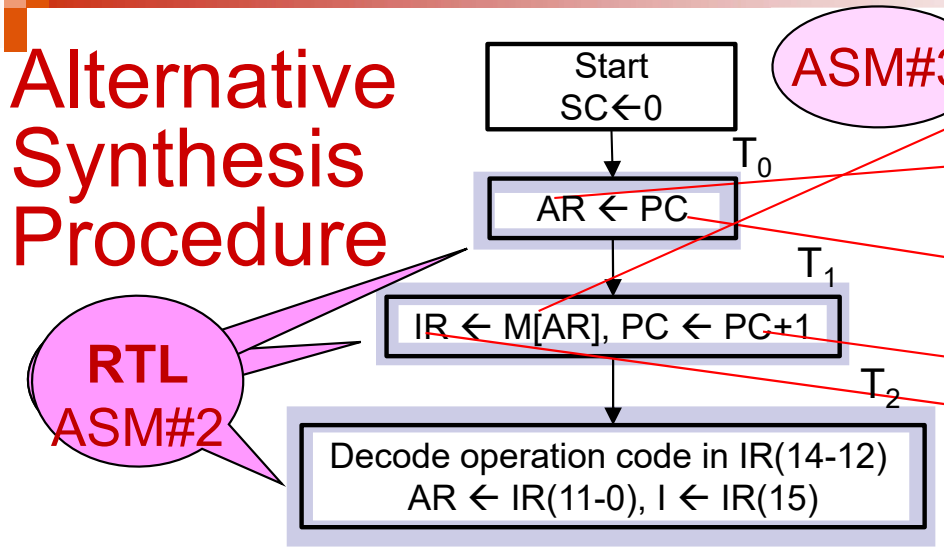
ASM#3

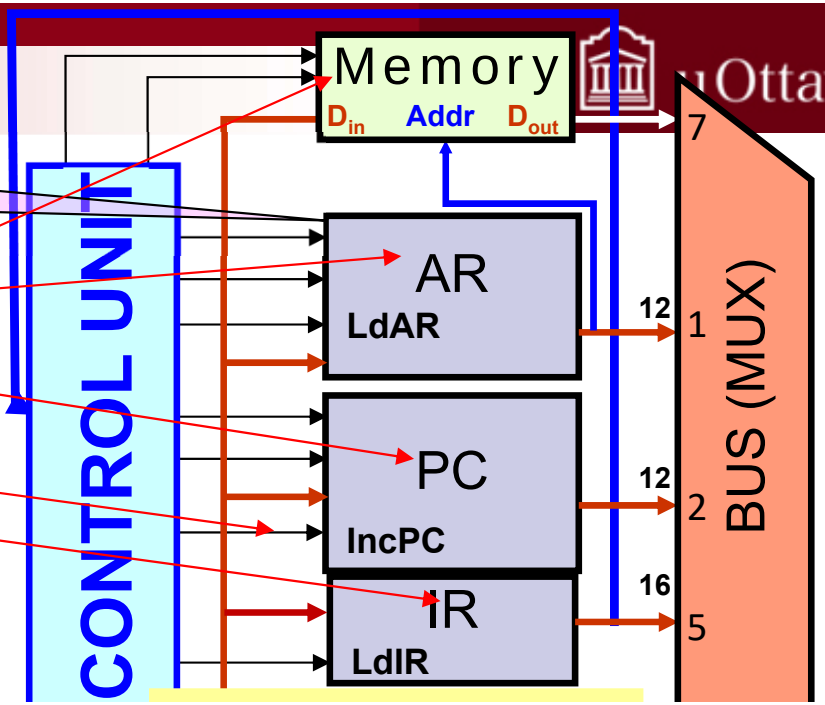
RTL Flowchart

ASM#5
Mano's approach

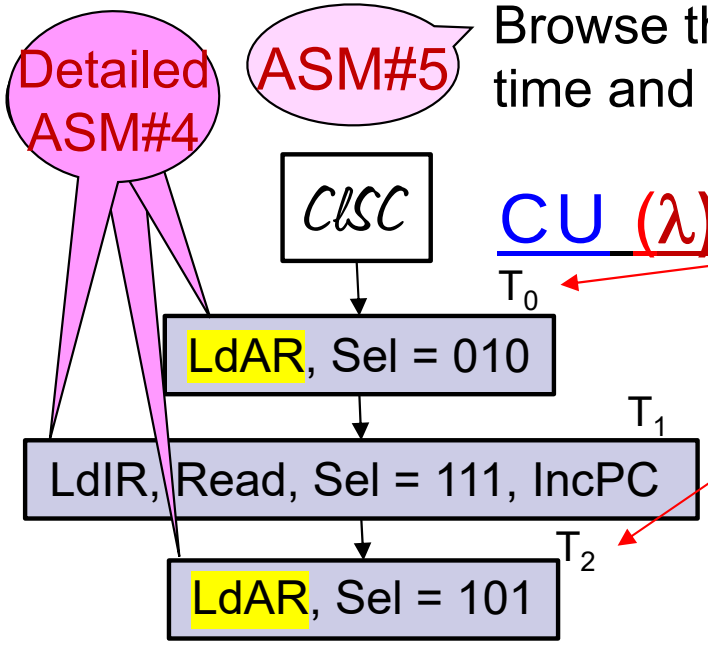


Alternative Synthesis Procedure





Instruction **FETCH**: Control Signals Synthesis & Timing



Browse the ASM detailed flowchart (ASM#4) for a control signal at a time and form its SoP with the states when it has to be generated.

CU (λ)

T_0

Let's start with LdAR:

$LdAR = T_0 + T_2 + \ldots$

Continue with the rest:

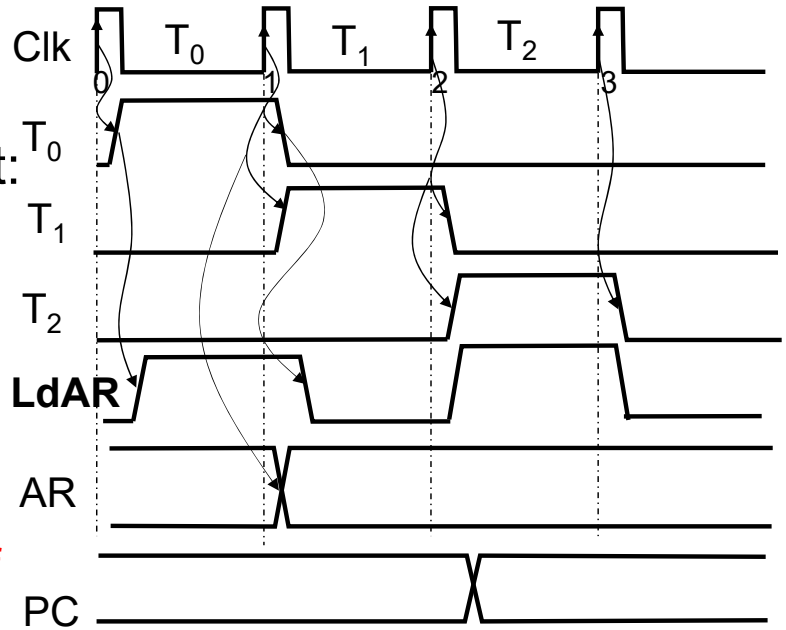
$LdIR = T_1 + \ldots$

$Read = T_1 + \ldots$

$Sel_2 = T_1 + T_2 + \ldots$

$Sel_1 = T_0 + T_1 \ldots$

$Sel_0 = T_1 + T_2 + \ldots$



However, one cannot finish the project here because of lack of specifications about the instructions executions!

All instructions detailed specifications of

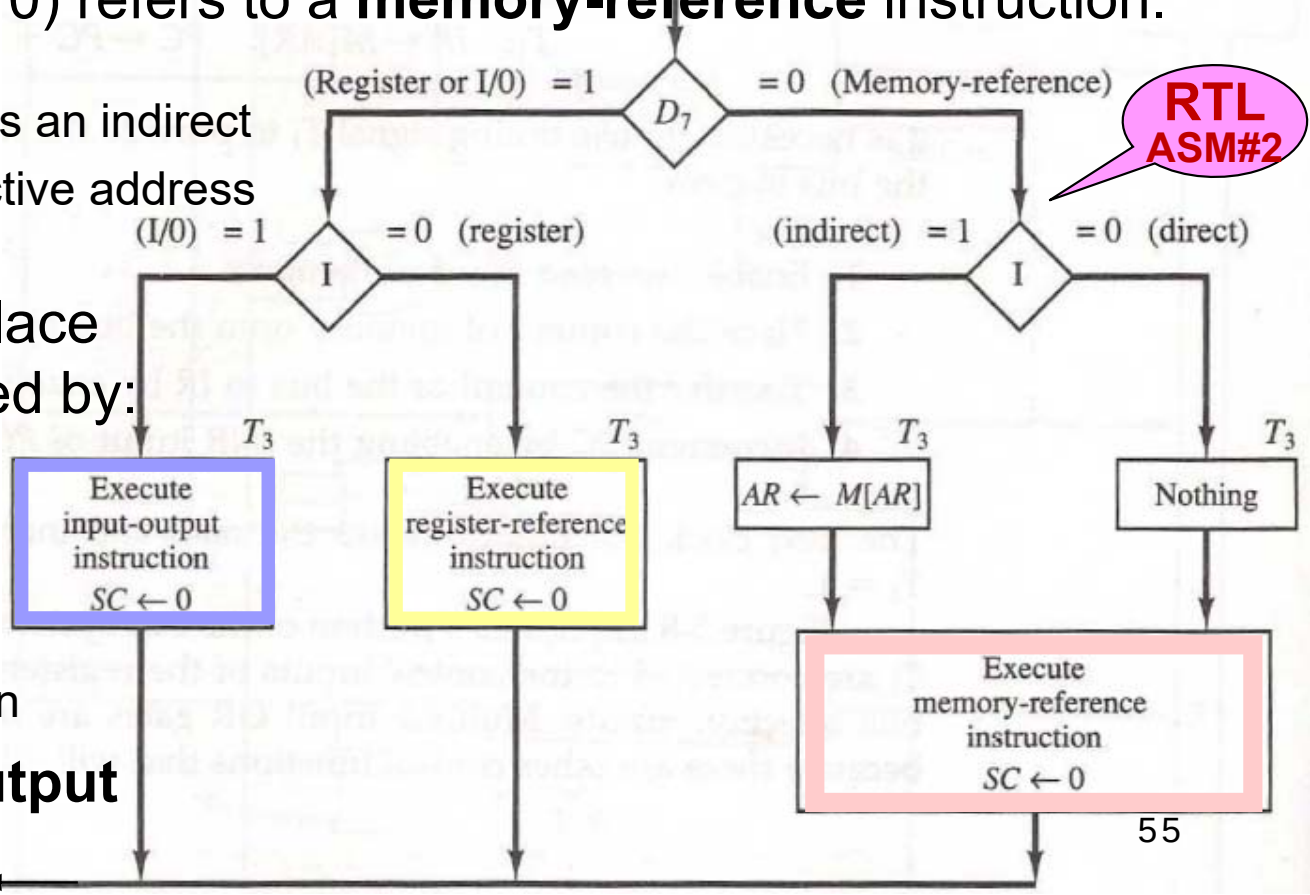
EXECUTION

in RTL ASM (ASM#2)

and detailed ASM (ASM#4)

Determine the Type of Instruction

- The instruction fetching and decoding take place at times T_0 to T_2.
- During T_3, CU determines the type of instruction just read from the memory.
- The following segment of the ASM flowchart for instruction cycle presents an initial configuration of the instruction cycle.
- D_7 = 0 (opcode is 000 to 110) refers to a **memory-reference** instruction.
  - If I = 1, then the instruction carries an indirect address and the operand's effective address is read by transferring it to AR.
- All four operations taking place at time T_3 can be symbolized by:
  - $D'_7\, I\, T_3$: $AR \leftarrow M[AR]$
  - $D'_7\, I'\, T_3$: Nothing
  - $D_7\, I'\, T_3$: Execute a **register-reference** instruction
  - $D_7\, I\, T_3$: Execute an **input-output** instruction



Basic Computer Instruction List (Binary)

D_7		I_{15}=0	I_{15}=1	op code			I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
0	*mem*	**MRI Direct Addr.**	**MRI** Indirect Addr.	*x*	*x*	*x*	Memory				A	D	D	R	E	S	S	
1	*reg*	**RRI**	**IOI**	**1**	**1**	**1**	C	o	d	e	Extension							

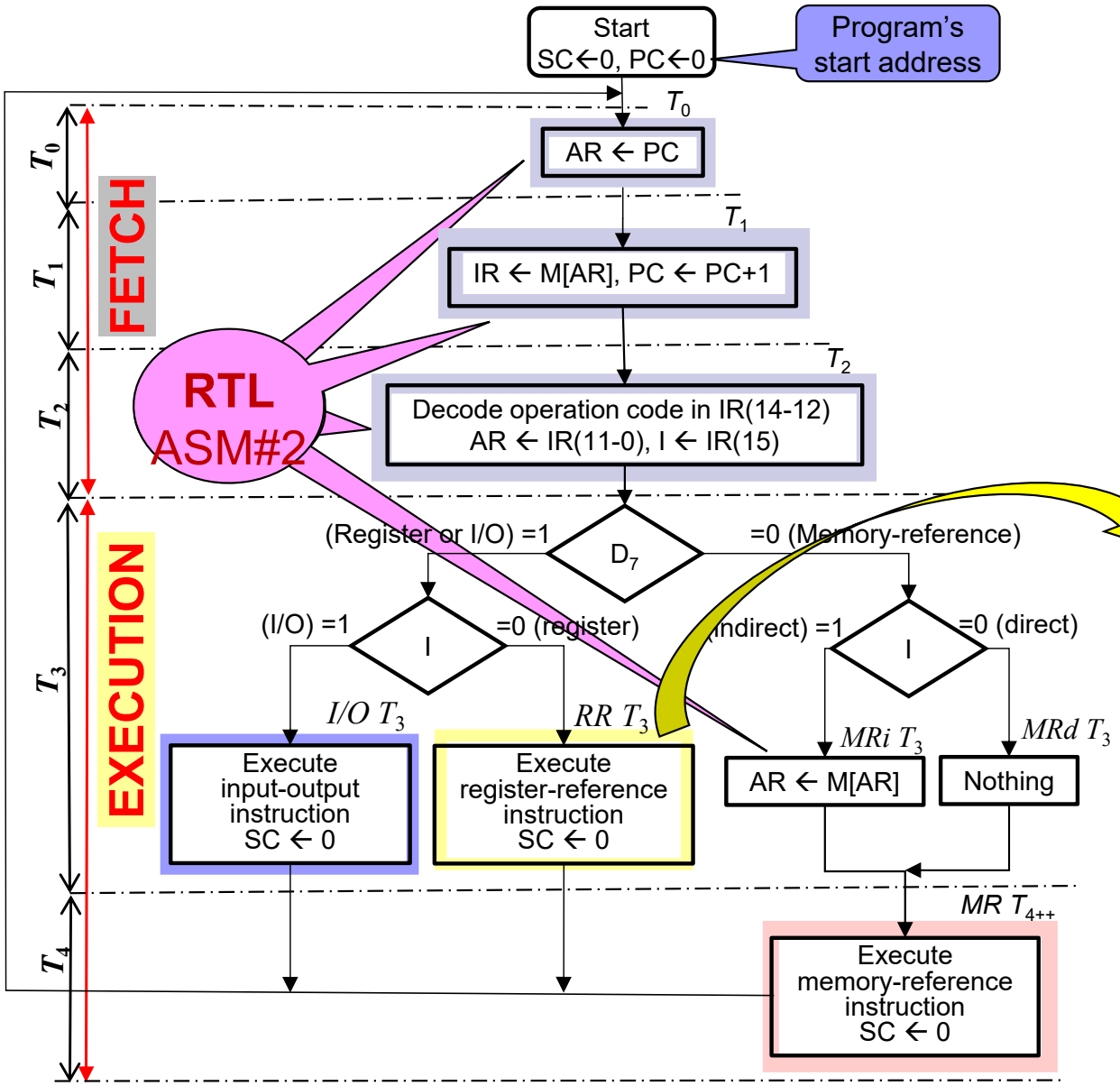
D_7	*Instr.* *Dec*	Addressing Mode I_{15}=0	I_{15}=1	op code I_{14}	I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
		MRI Direct Addr.	**MRI** Indirect Addr.				Memory				A	D	D	R	E	S	S	
D_7=0 ***mem***	D_0	**AND**=$0addr	AND_i =$8(addr)	*0*	*0*	*0*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_1	**ADD**=$1addr	ADD_i =$9(addr)	*0*	*0*	*1*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_2	**LDA** =$2addr	LDA_i =$A(addr)	*0*	*1*	*0*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_3	**STA** =$3addr	STA_i =$B(addr)	*0*	*1*	*1*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_4	**BUN** =$4addr	BUN_i =$C(addr)	*1*	*0*	*0*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_5	**BSA** =$5addr	BSA_i =$D(addr)	*1*	*0*	*1*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
	D_6	**ISZ** =$6addr	ISZ_i =$E(addr)	*1*	*1*	*0*	AR_{11}	AR_{10}	AR_9	AR_8	AR_7	AR_6	AR_5	AR_4	AR_3	AR_2	AR_1	AR_0
		RRI	**IOI**	**1**	**1**	**1**	C	o	d	e	Extension							
D_7=1 ***reg***	D_7	**CLA** =$7800	INP = $F800	*1*	*1*	*1*	**1**	0	0	0	0	0	0	0	0	0	0	0
	D_7	**CLE** =$7400	OUT = $F400	*1*	*1*	*1*	0	**1**	0	0	0	0	0	0	0	0	0	0
	D_7	**CMA** =$7200	SKI = $F200	*1*	*1*	*1*	0	0	**1**	0	0	0	0	0	0	0	0	0
	D_7	**CME** =$7100	SKO = $F100	*1*	*1*	*1*	0	0	0	1	0	0	0	0	0	0	0	0
	D_7	**CIR** =$7080	ION = $F080	*1*	*1*	*1*	0	0	0	0	**1**	0	0	0	0	0	0	0
	D_7	**CIL** =$7040	IOF = $F040	*1*	*1*	*1*	0	0	0	0	0	**1**	0	0	0	0	0	0
	D_7	**INC** =$7020	n/a	*1*	*1*	*1*	0	0	0	0	0	0	**1**	0	0	0	0	0
	D_7	**SPA** =$7010	n/a	*1*	*1*	*1*	0	0	0	0	0	0	0	**1**	0	0	0	0
	D_7	**SNA** =$7008	n/a	*1*	*1*	*1*	0	0	0	0	0	0	0	0	**1**	0	0	0
	D_7	**SZA** =$7004	n/a	*1*	*1*	*1*	0	0	0	0	0	0	0	0	0	**1**	0	0
	D_7	**SZE** =$7002	n/a	*1*	*1*	*1*	0	0	0	0	0	0	0	0	0	0	**1**	0
	D_7	**HLT** =$7001	n/a	*1*	*1*	*1*	0	0	0	0	0	0	0	0	0	0	0	**1**

Binary encoding | **One-hot** (*bit-per-state*) encoding

"addr" = 12 bit address of the operand
"(addr)" = address of the operand address

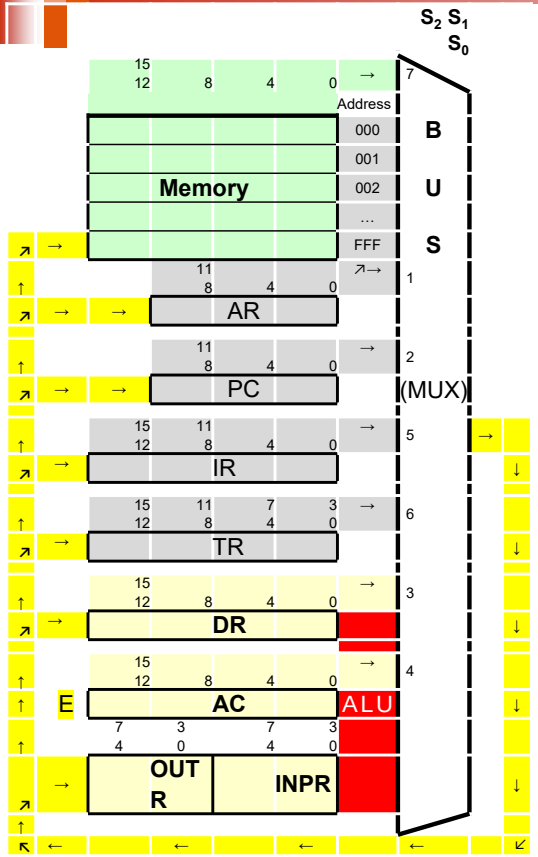
uOttawa

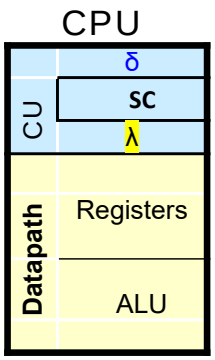
Control Functions & Microoperations of the Basic Computer



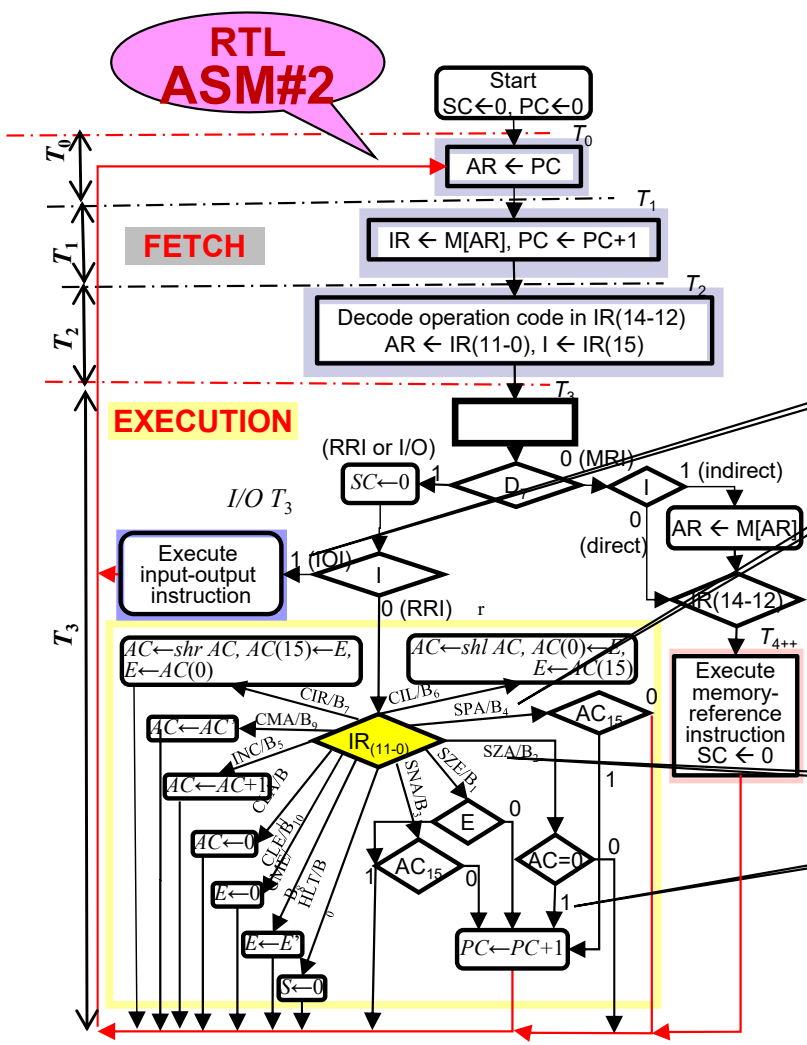
Fetch	T_o:	$AR \leftarrow PC$
	T_1:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	T_2:	$D_0, \ldots, D_7 \leftarrow$ Decode IR(12-14), $AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Memory-reference:		
AND		
ADD		
LDA		
STA.		
BUN		
BSA		
ISZ		
Reg-ref		
CLA		
CLE		
CMA		
CME		
CIR		
CIL		
INC		
SPA		
SNA		
SZA		
SZE		
HLT		
In-out		
INP		
OUT		
SKI		
SKO		
ION		
IOF		

Execution of Register Reference Instructions





The CU can be implemented simply with a binary counter (called Sequence Counter SC), which goes through the sequence of states $T_0, T_1, T_2 \ldots$ and returns to T_0 after finalizing execution of the instruction.



RTL

Fetch	T_0:	$AR \leftarrow PC$
	T_1:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	T_2:	$D_0, \ldots, D_7 \leftarrow Decode\ IR(12\text{-}14), AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$
Indirect	$D'_7 I T_3$:	$AR \leftarrow M[AR]$
Memory-Ref.		
Reg-ref	$D_7 I' T_3 = r$	(common to all reg.-ref. instr.)
	$IR(i) = B_i$	(i = 0, 1, 2, ... , 11)
	r:	$SC \leftarrow 0$
Clear AC	CLA rB_{11}:	$AC \leftarrow 0$
Clear E	CLE rB_{10}:	$E \leftarrow 0$
Complement AC	CMA rB_9:	$AC \leftarrow AC'$
Complement E	CME rB_8:	$E \leftarrow E'$
Circulate right AC and E	CIR rB_7:	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
Circulate left AC and E	CIL rB_6:	$AC \leftarrow shl AC, AC(0) \leftarrow E, E \leftarrow AC(15)$
Increment AC	INC rB_5:	$AC \leftarrow AC + 1$
Skip next instruction if AC positive	SPA rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC+1)$
Skip next instruction if AC negative	SNA rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC+1)$
Skip next instruction if AC zero	SZA rB_2:	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$
Skip next instruction if E is 0	SZE rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
Halt computer	HLT rB_0:	$S \leftarrow 0$
In-out		

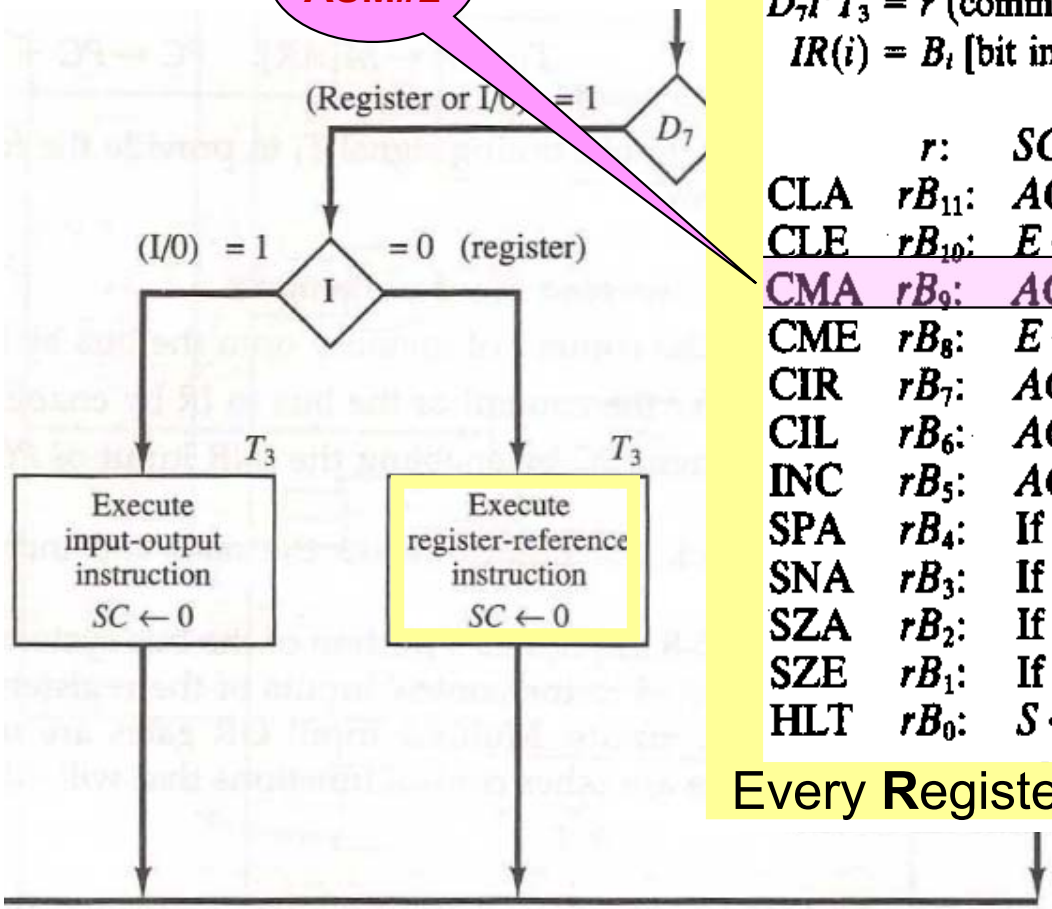
Instruction Cycle (EXEC)
Register-Reference Instructions (RRI)

Opcode from the Instruction List: $I_{14}I_{13}I_{12} = 111 \Rightarrow D_7 = 1$

$I_{15} = I = 0$

register

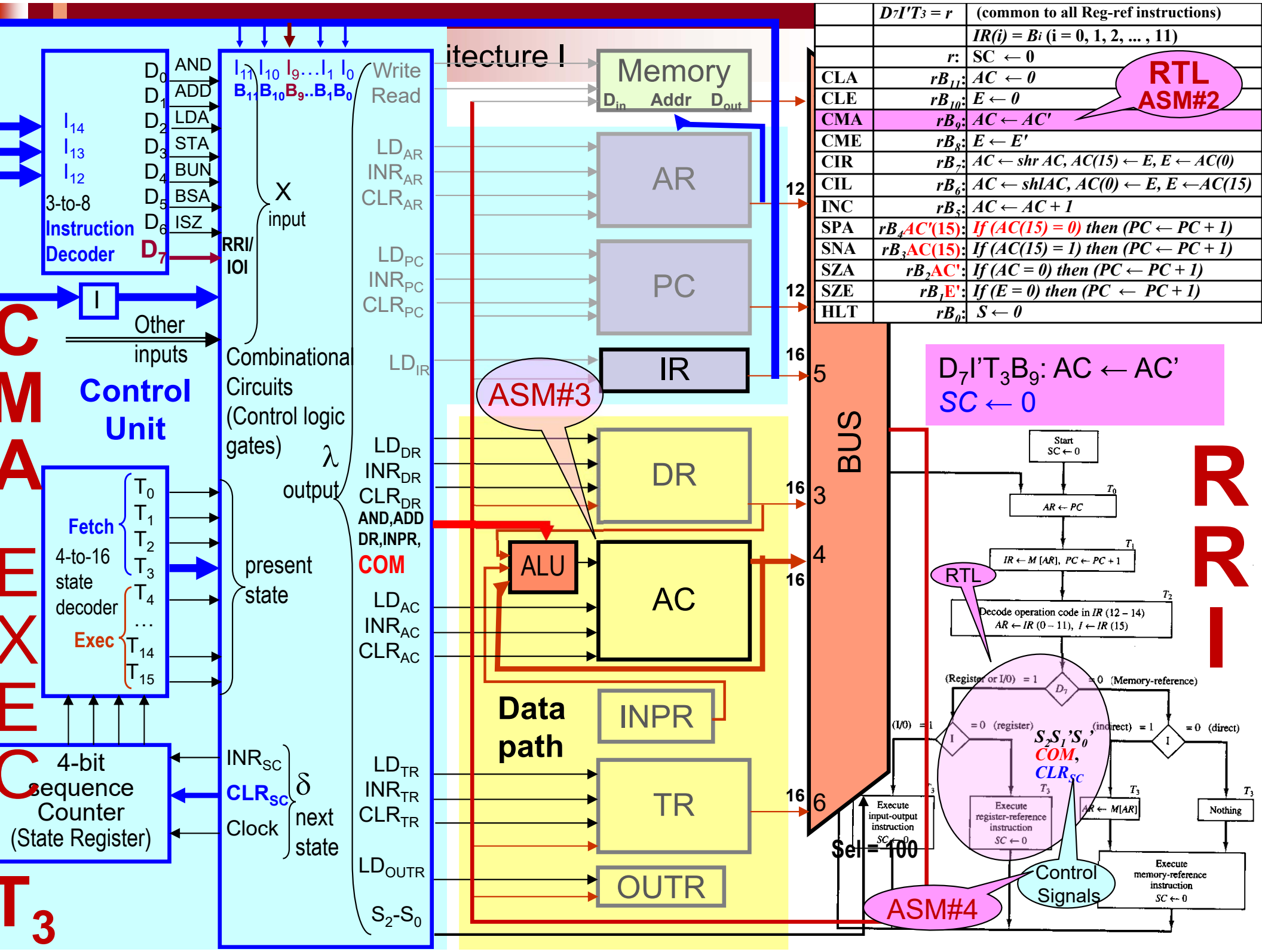
RTL ASM#2



$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0\text{–}11)$ that specifies the operation]

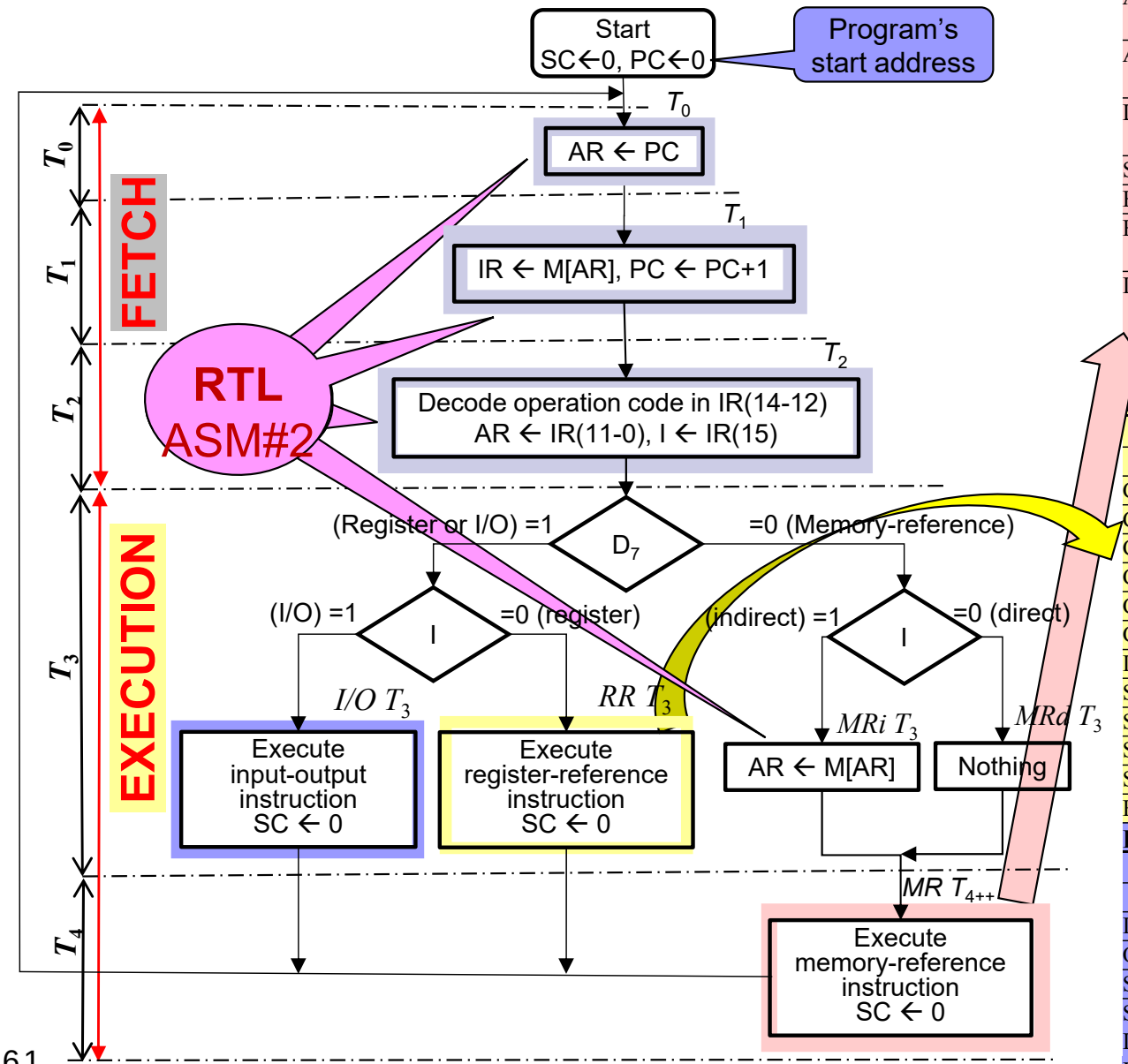
	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

Every **R**egister-**R**eference **I**nstruction is executed during T_3


itecture I
Memory
D_{in} Addr D_{out}
AR
PC
IR
DR
ALU
AC
INPR
TR
OUTR
BUS
12
12
16
5
16
3
4
16
16
6
Write
Read
LD_{AR} INR_{AR} CLR_{AR}
LD_{PC} INR_{PC} CLR_{PC}
LD_{IR}
LD_{DR} INR_{DR} CLR_{DR}
AND,ADD
DR,INPR,
COM
LD_{AC} INR_{AC} CLR_{AC}
LD_{TR} INR_{TR} CLR_{TR}
LD_{OUTR}
S_2-S_0
I_{14} I_{13} I_{12}
3-to-8
Instruction
Decoder
D_0 AND
D_1 ADD
D_2 LDA
D_3 STA
D_4 BUN
D_5 BSA
D_6 ISZ
D_7
RRI/
IOI
I_{11} I_{10} I_9…I_1 I_0
$B_{11}B_{10}B_9..B_1B_0$
X
input
I
Other
inputs
Combinational
Circuits
(Control logic
gates)
λ
output
Control
Unit
Fetch
T_0 T_1 T_2 T_3
4-to-16
state
decoder
Exec
T_4 … T_{14} T_{15}
present
state
4-bit
Sequence
Counter
(State Register)
INR_{SC}
CLR_{SC}
Clock
δ
next
state
ASM#3
Data
path
CMA
EXEC
T_3
RRI
$D_7I'T_3 = r$ (common to all Reg-ref instructions)
$IR(i) = B_i$ (i = 0, 1, 2, ... , 11)
r: SC ← 0
CLA rB_{11}: AC ← 0
CLE rB_{10}: E ← 0
CMA rB_9: AC ← AC'
CME rB_8: E ← E'
CIR rB_7: AC ← shr AC, AC(15) ← E, E ← AC(0)
CIL rB_6: AC ← shlAC, AC(0) ← E, E ←AC(15)
INC rB_5: AC ← AC + 1
SPA rB_4 AC'(15): If (AC(15) = 0) then (PC ← PC + 1)
SNA rB_3 AC(15): If (AC(15) = 1) then (PC ← PC + 1)
SZA rB_2 AC': If (AC = 0) then (PC ← PC + 1)
SZE rB_1 E': If (E = 0) then (PC ← PC + 1)
HLT rB_0: S ← 0
RTL
ASM#2
$D_7I'T_3B_9$: AC ← AC'
SC ← 0
Start
SC ← 0
T_0
AR ← PC
T_1
IR ← M [AR], PC ← PC + 1
RTL
T_2
Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)
(Register or I/0) = 1
D_7
= 0 (Memory-reference)
(I/0) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
$S_2S_1'S_0'$
COM,
CLR_{SC}
Execute
input-output
instruction
SC ← 0
Execute
register-reference
instruction
SC ← 0
AR ← M[AR]
Nothing
Execute
memory-reference
instruction
SC ← 0
Sel = 100
Control
Signals
ASM#4

uOttawa

Control Functions & Microoperations of the Basic Computer



Fetch	T_0:	$AR \leftarrow PC$
	T_1:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	T_2:	$D_0, \ldots, D_7 \leftarrow Decode\ IR(12\text{-}14), AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Memory-reference:		
AND		
ADD		
LDA		
STA.		
BUN		
BSA		
ISZ		
Reg-ref	$D_7I'T_3 = r$	(common to all reg.-ref. instr.) $IR(i) = B_i$ (i = 0, 1, 2, ... , 11)
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shlAC, AC(0) \leftarrow E, E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	$rB_4AC'(15)$:	$If\ (AC(15) = 0)\ then\ (PC \leftarrow PC + 1)$
SNA	$rB_3AC(15)$:	$If\ (AC(15) = 1)\ then\ (PC \leftarrow PC + 1)$
SZA	rB_2AC':	$If\ (AC = 0)\ then\ (PC \leftarrow PC + 1)$
SZE	rB_1E':	$If\ (E = 0)\ then\ (PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
In-out		
INP		
OUT		
SKI		
SKO		
ION		
IOF		

Instruction Cycle (EXEC)
Memory-Reference Instructions (MRI)

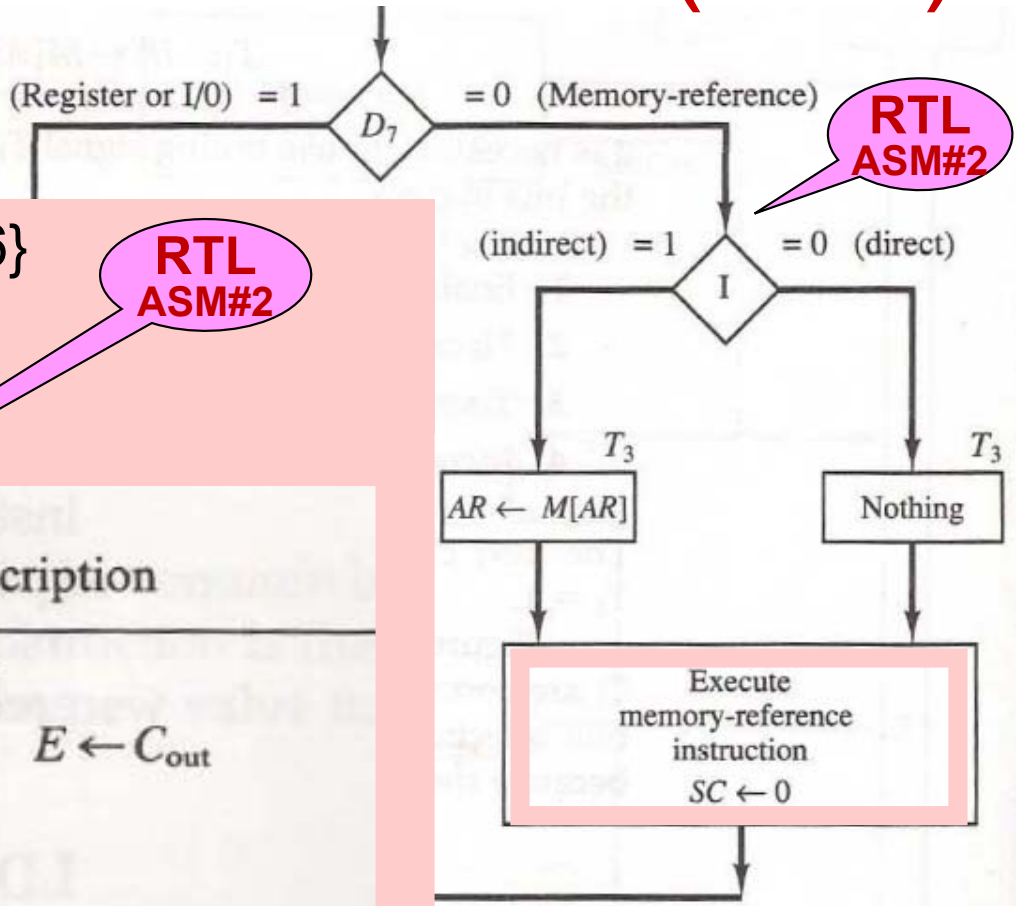
Opcodes from the Instruction List:

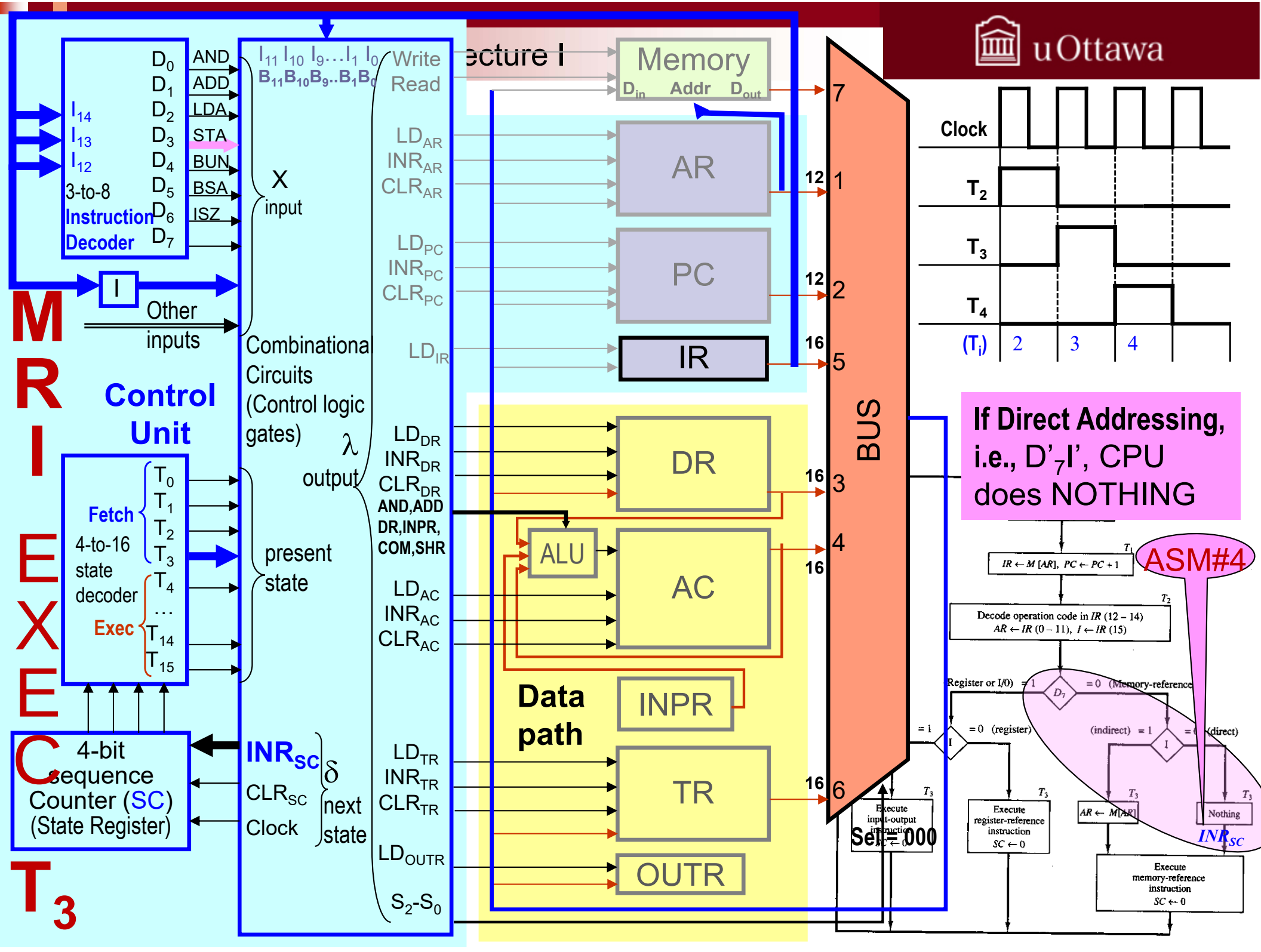
$I_{14}I_{13}I_{12}$ = [000 – 110] => $D_i = 1$, i = {0 .. 6}

- $I_{15} = I = 0$ => direct addressing
- $I_{15} = I = 1$ => indirect addressing

RTL ASM#2

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$



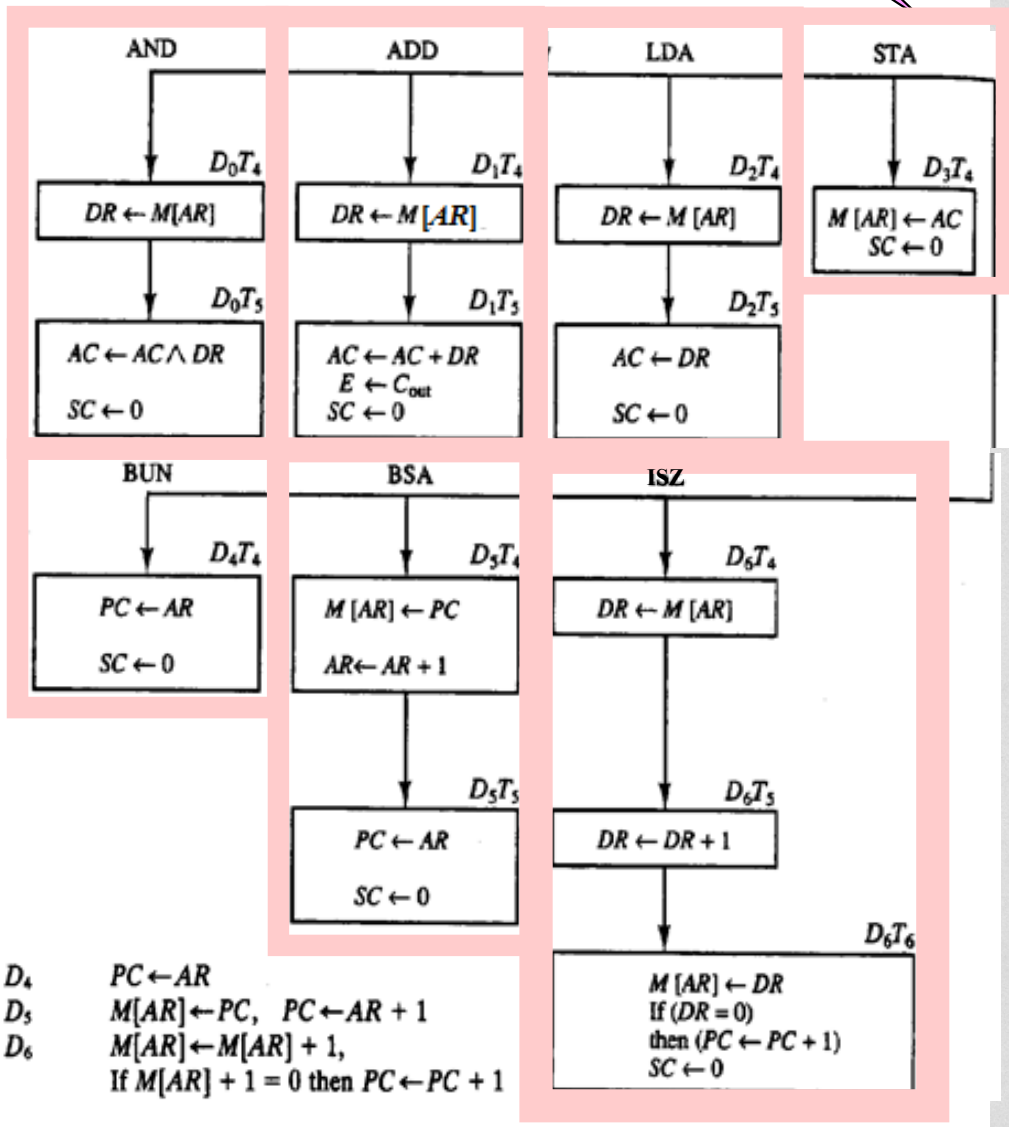

ecture I
Memory
Din Addr Dout
AR
PC
IR
DR
ALU
AC
INPR
TR
OUTR
BUS
7
12
1
12
2
16
5
16
3
4
16
16
6
Clock
T2
T3
T4
(Ti) 2 3 4
I14
I13
I12
3-to-8
Instruction
Decoder
D0 AND
D1 ADD
D2 LDA
D3 STA
D4 BUN
D5 BSA
D6 ISZ
D7
I
Other
inputs
I11 I10 I9...I1 I0
B11B10B9..B1B0
X
input
Write
Read
LDAR
INRAR
CLRAR
LDPC
INRPC
CLRPC
LDIR
Combinational
Circuits
(Control logic
gates)
λ
output
LDDR
INRDR
CLRDR
AND,ADD
DR,INPR,
COM,SHR
LDAC
INRAC
CLRAC
LDTR
INRTR
CLRTR
LDOUTR
S2-S0
Control
Unit
Fetch
4-to-16
state
decoder
Exec
T0
T1
T2
T3
T4
...
T14
T15
present
state
4-bit
Sequence
Counter (SC)
(State Register)
INRSC
CLRSC
Clock
δ
next
state
Data
path
MRI
EXEC
T3
If Direct Addressing,
i.e., D'7I', CPU
does NOTHING
ASM#4
IR ← M [AR], PC ← PC + 1
Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)
Register or I/0) = 1
D7
= 0 (Memory-reference
= 1
I
= 0 (register)
(indirect) = 1
I
(direct)
Execute
input-output
instruction
SC ← 0
Execute
register-reference
instruction
SC ← 0
AR ← M[AR]
Nothing
INRSC
Execute
memory-reference
instruction
SC ← 0
Sel = 000

[AR]

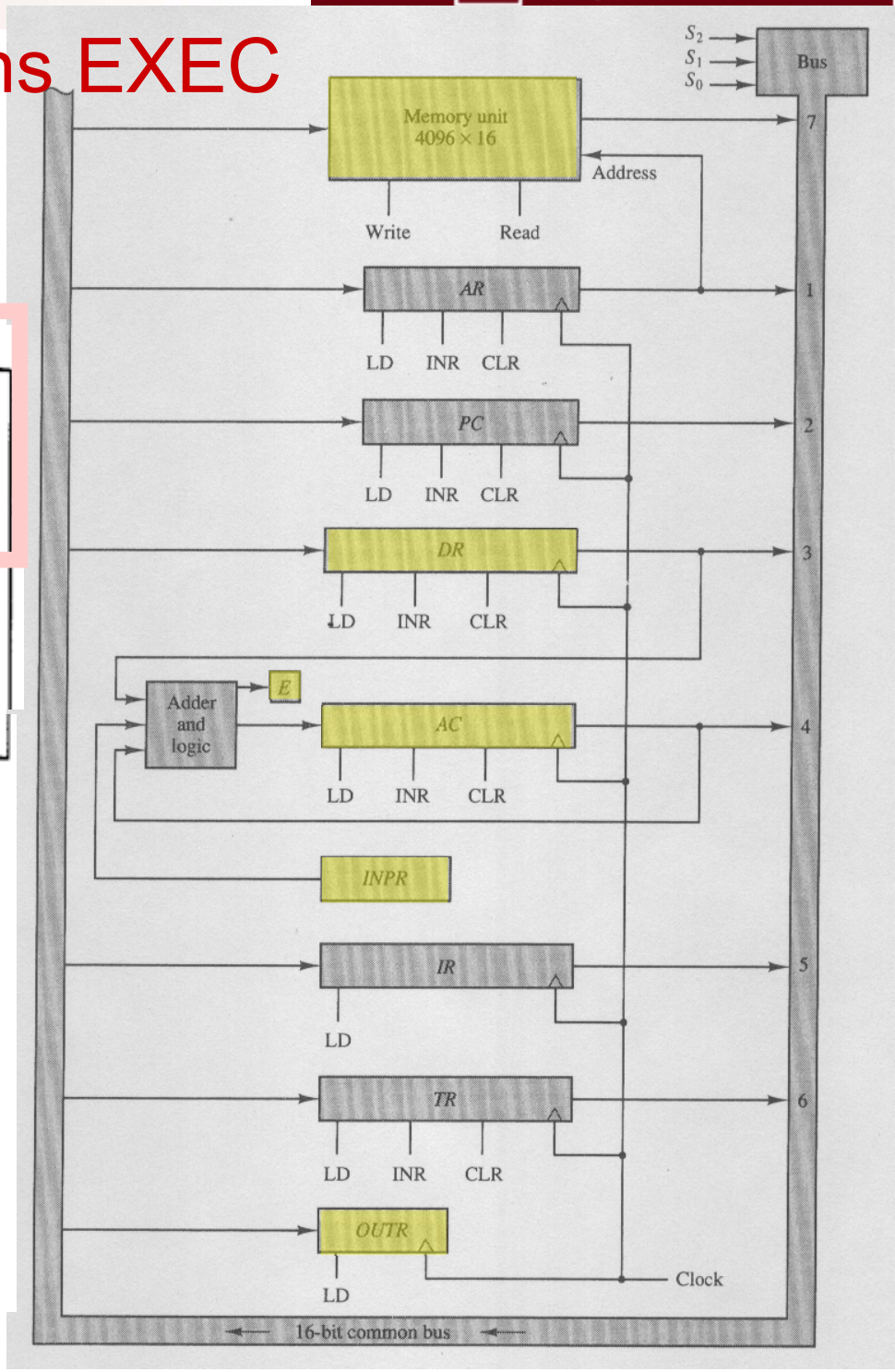
Memory-Reference Instructions EXEC

RTL ASM#2

AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR]$, $E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$



BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC$, $PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

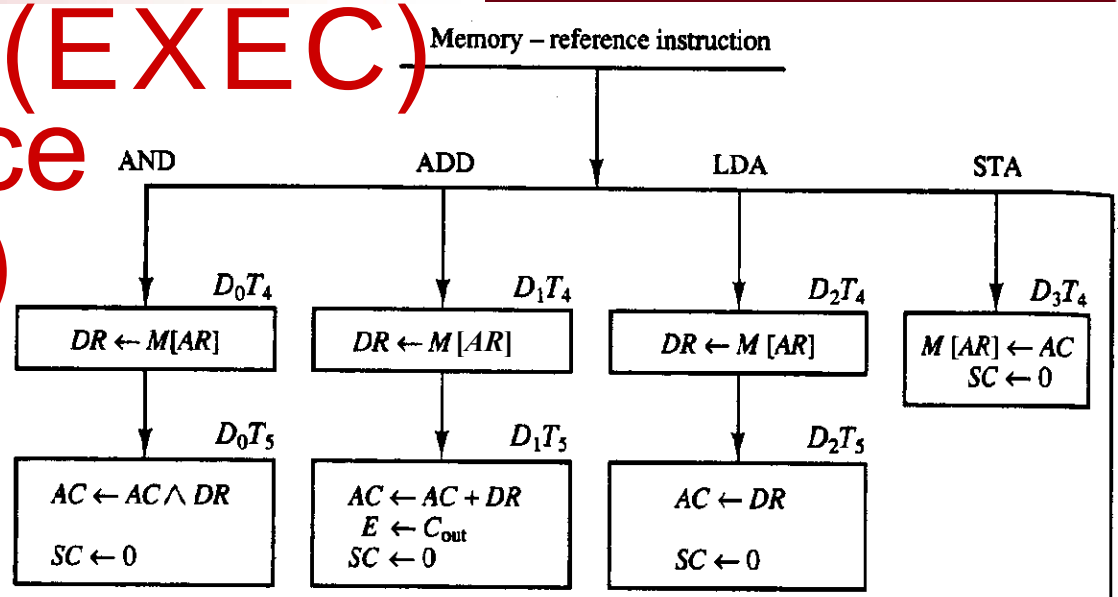


Instruction Cycle (EXEC) Memory-Reference Instructions (MRI)

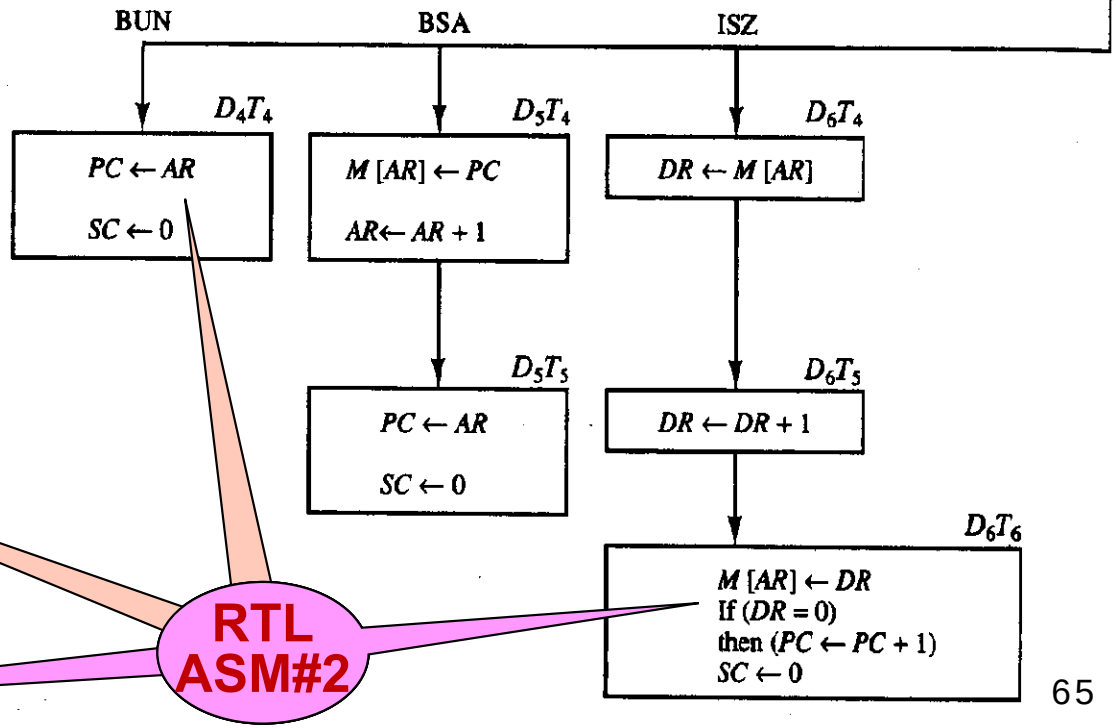
Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR]$, $E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC$, $PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

ASM#1

AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR$, $SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR$, $SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, $SC \leftarrow 0$
	D_6T_6DR':	if $(DR = 0)$ then $(PC \leftarrow PC + 1)$



← From the Instruction List:



RTL ASM#2

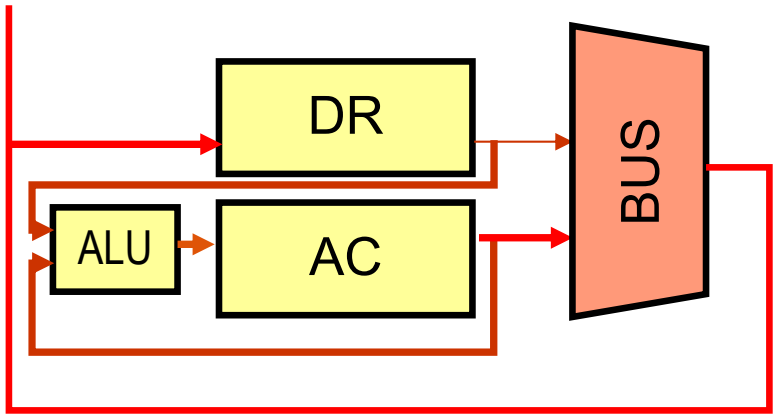
Note on RTL

- A RTL statement consists in an (optional) condition followed by micro-operation(s).
- The micro-operation should be read as having the present values on the right side of "←" , while the left side refers to the next state of that register; the transfer (register loading) takes place on the rising edge of the clock.
- For example:

D_1T_5: DR ← AC, AC ← AC + DR

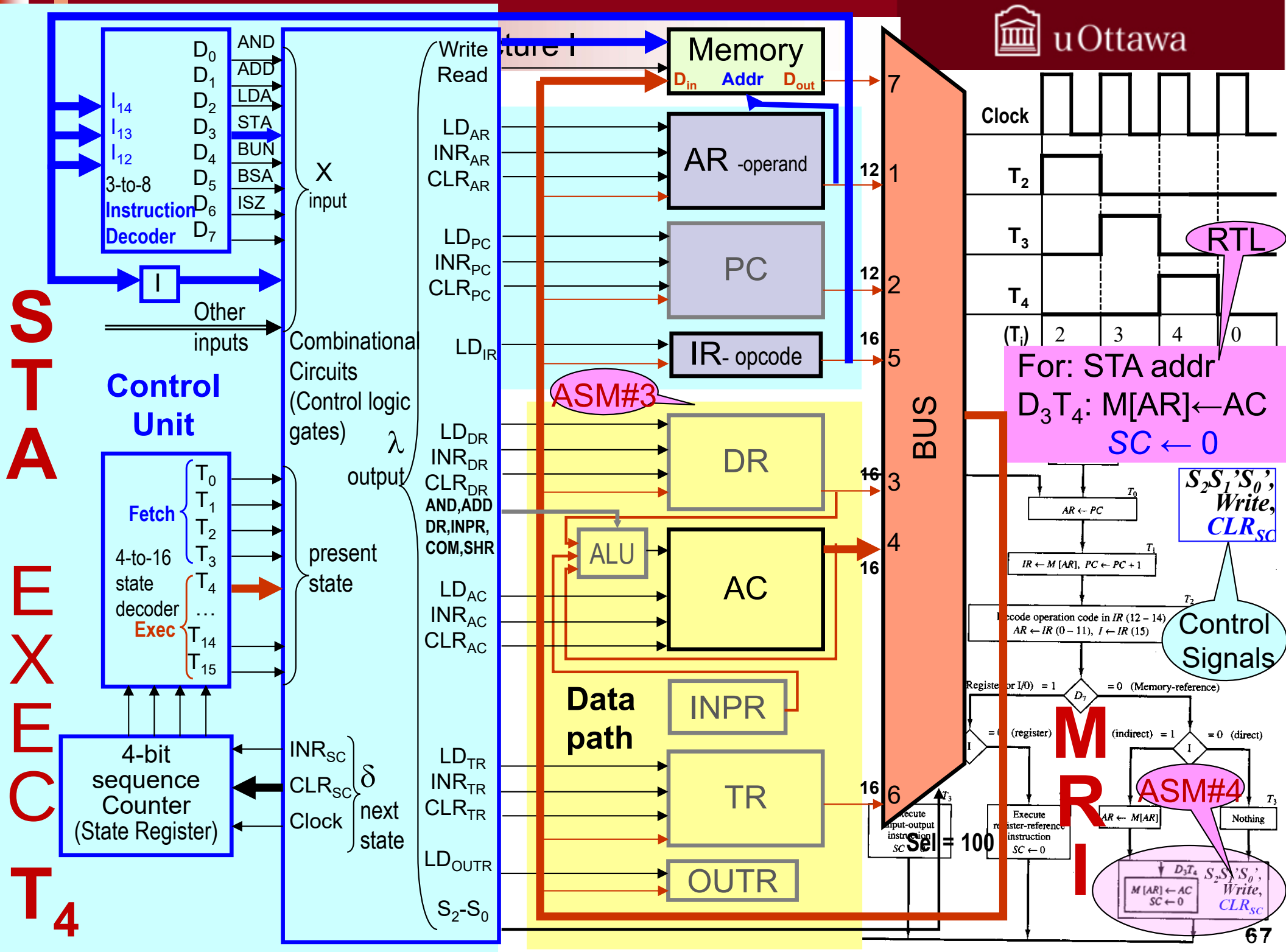
actually reads:

D_1T_5: DR(n+1)← AC(n)
AC(n+1)← AC(n) + DR(n)



Even the second RTL makes sense since **AC(n)+DR(n)** is the ALU's output which is to be loaded into AC on the rising edge of the clock n+1. After loading the newer value into AC, there will be a delay until the ALU output will change, since ALU needs some time to "figure out" what is the newer result. As such, by the time ALU output will change, the rising edge will be gone and nothing will be allowed into AC until the next rising edge!

STA EXEC T_4


Memory
Din Addr Dout
AR -operand
PC
IR- opcode
DR
ALU
AC
INPR
TR
OUTR
BUS
7
12 1
12 2
16 5
16 3
4
16
16 6
Write
Read
LDAR INRAR CLRAR
LDPC INRPC CLRPC
LDIR
LDDR INRDR CLRDR
AND,ADD
DR,INPR,
COM,SHR
LDAC INRAC CLRAC
LDTR INRTR CLRTR
LDOUTR
S2-S0
Data path
ASM#3
D0 AND
D1 ADD
D2 LDA
D3 STA
D4 BUN
D5 BSA
D6 ISZ
D7
I14 I13 I12
3-to-8 Instruction Decoder
I
Other inputs
X input
Combinational Circuits (Control logic gates)
λ output
Control Unit
T0 T1 T2 T3 Fetch
T4 … T14 Exec
4-to-16 state decoder
present state
INRSC CLRSC Clock
δ next state
4-bit sequence Counter (State Register)
Clock
T2
T3
T4
(Ti) 2 3 4 0
RTL
For: STA addr
D3T4: M[AR]←AC
SC ← 0
S2S1'S0', Write, CLRSC
Control Signals
AR ← PC
IR ← M [AR], PC ← PC + 1
Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)
(Register or I/O) = 1
D7
= 0 (Memory-reference)
I
(register)
(indirect) = 1
= 0 (direct)
Execute input-output instruction SC ← 0
Execute register-reference instruction SC ← 0
AR ← M[AR]
Nothing
M R I
ASM#4
D3T4 S2S1'S0', Write, CLRSC
M [AR] ← AC SC ← 0
Sel = 100

uOttawa

ISZ – Increment & Skip if Zero implements DO-WHILE

DO

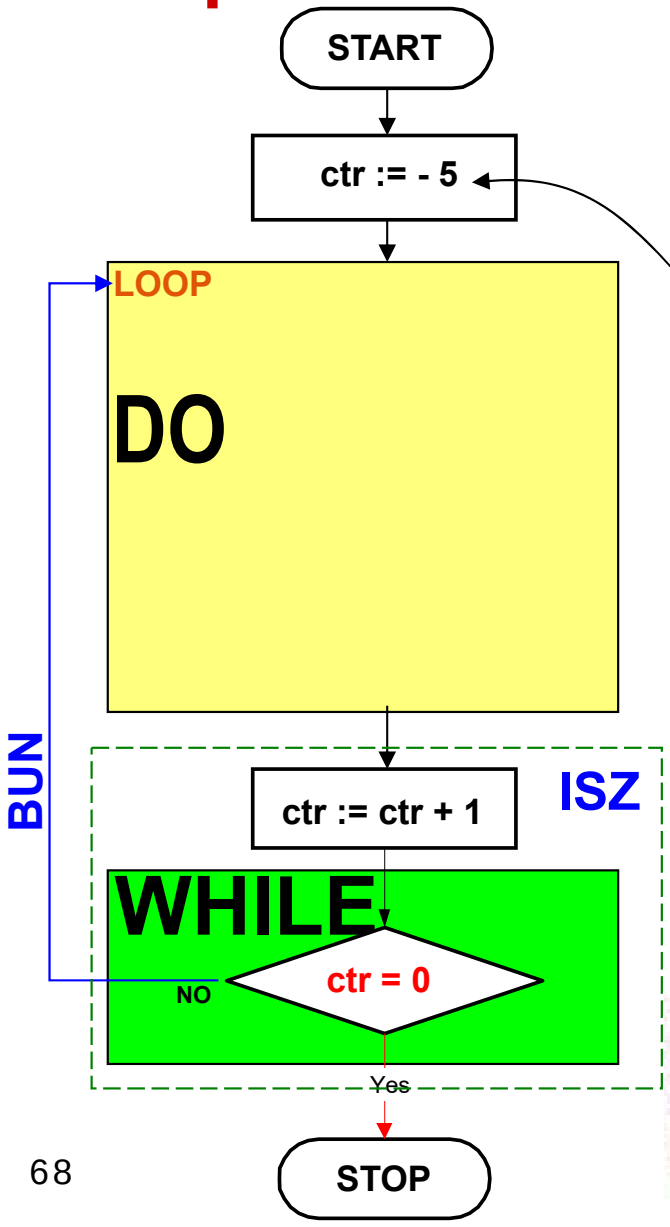
....

....

Increment ctr

ENDO

WHILE ctr < 0



MACHINE CODE		ASSEMBLY		
addr	memory			
000	...	LOOP	...	
001	...		...	
002	...		...	
003	6080		ISZ	CTR
004	4000		BUN	LOOP
005	7001		HLT	
...	...		...	
080	FFFB	CTR	equ -5	

if ctr = 0

MRI

ASM#2

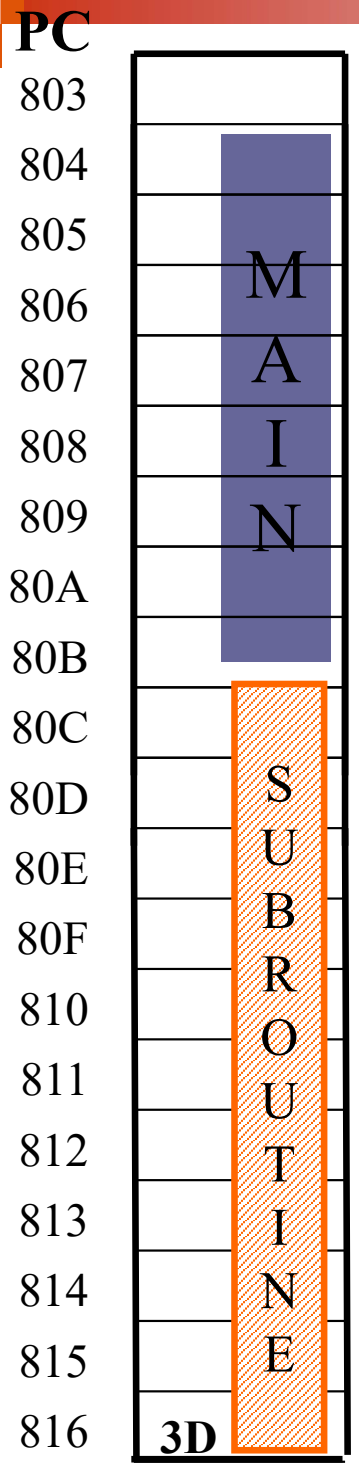
D_6T_4: $DR \leftarrow M[AR]$; $DR <= M[080] = FFFB$

D_6T_5: $DR \leftarrow DR + 1$; $DR <= FFFB + 1 = FFFC$

D_6T_6: $M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

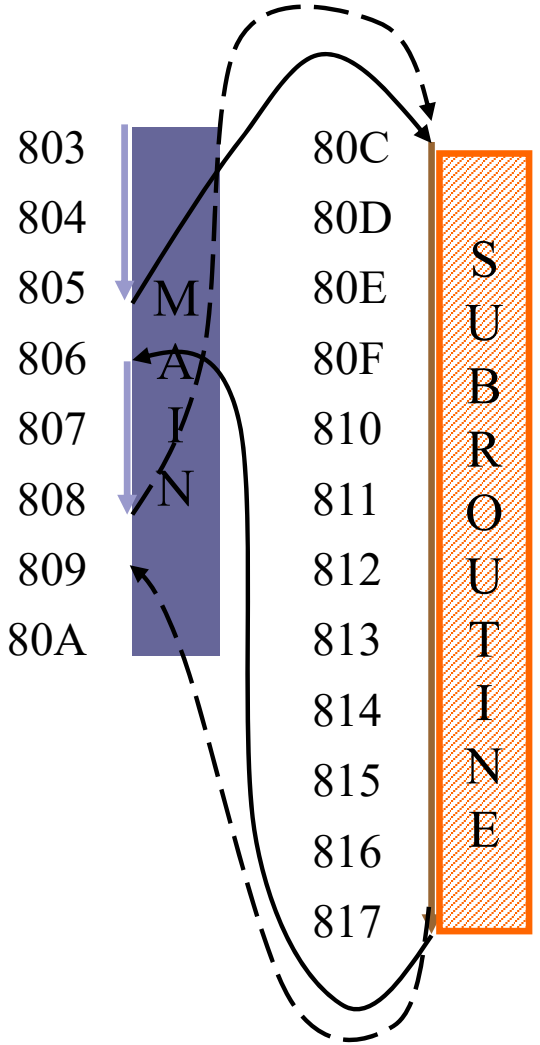
How …

1. … DOES A SUBROUTINE WORK?
2. … TO DEFINE IT?
3. … TO USE IT?



HOW DOES IT WORK?

- A subroutine is a program module that is independent of the main program
- To use it, the main program transfers control to the subroutine
- The subroutine performs its function and then returns control to the main program
- Main problem: define a mechanism to store the **RETURN ADDRESS**, such that the subroutine knows where to find it when it gets back to the calling program.

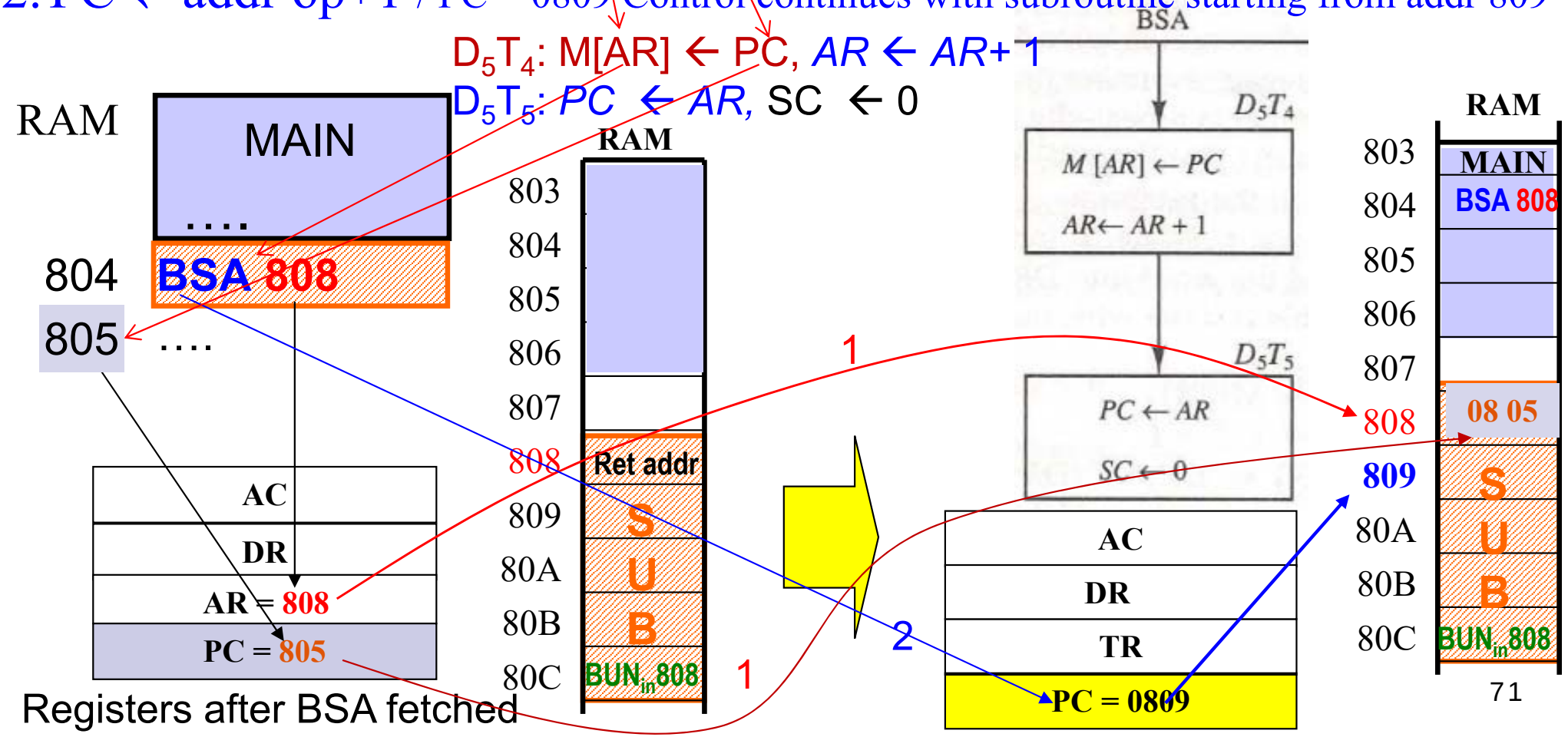


MAIN **CALLS** SUBROUTINE (**BSA**)

The **RETURN ADDRESS** is stored at the memory location pointed to by **BSA**. The subroutine starts at the following address.

1. m(addr op)←(PC) / m(808) = 0805 =return address **805** is stored in location 808
2. PC ←addr op+1 / PC = 0809 Control continues with subroutine starting from addr 809

D_5T_4: M[AR] ← PC, *AR* ← *AR*+ 1
D_5T_5: *PC* ← *AR*, SC ← 0

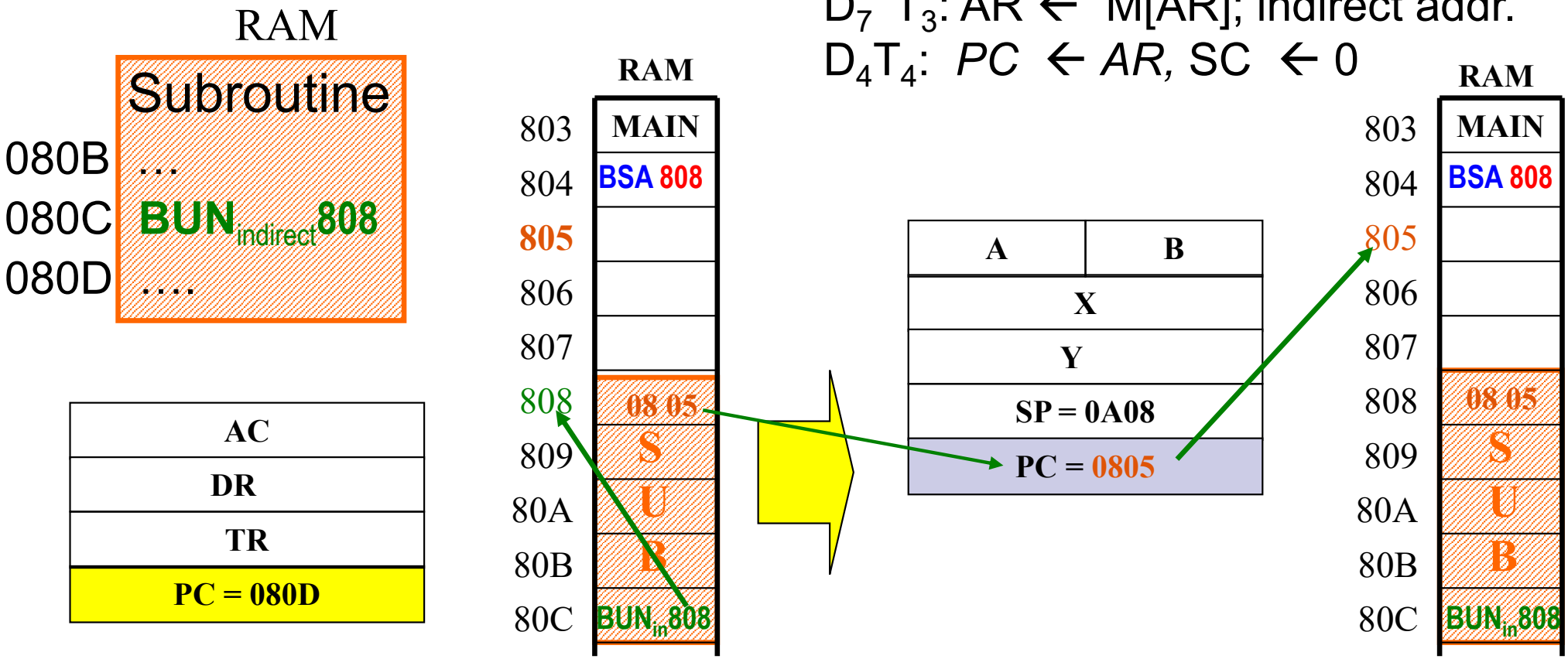


SUBROUTINE RETURNS TO MAIN

$BUN_{indirect}$ 808:

3. PC ← [m(m(808))]; PC=0805

$D_7'T_3$: AR ← M[AR]; indirect addr.
D_4T_4: *PC* ← *AR*, SC ← 0



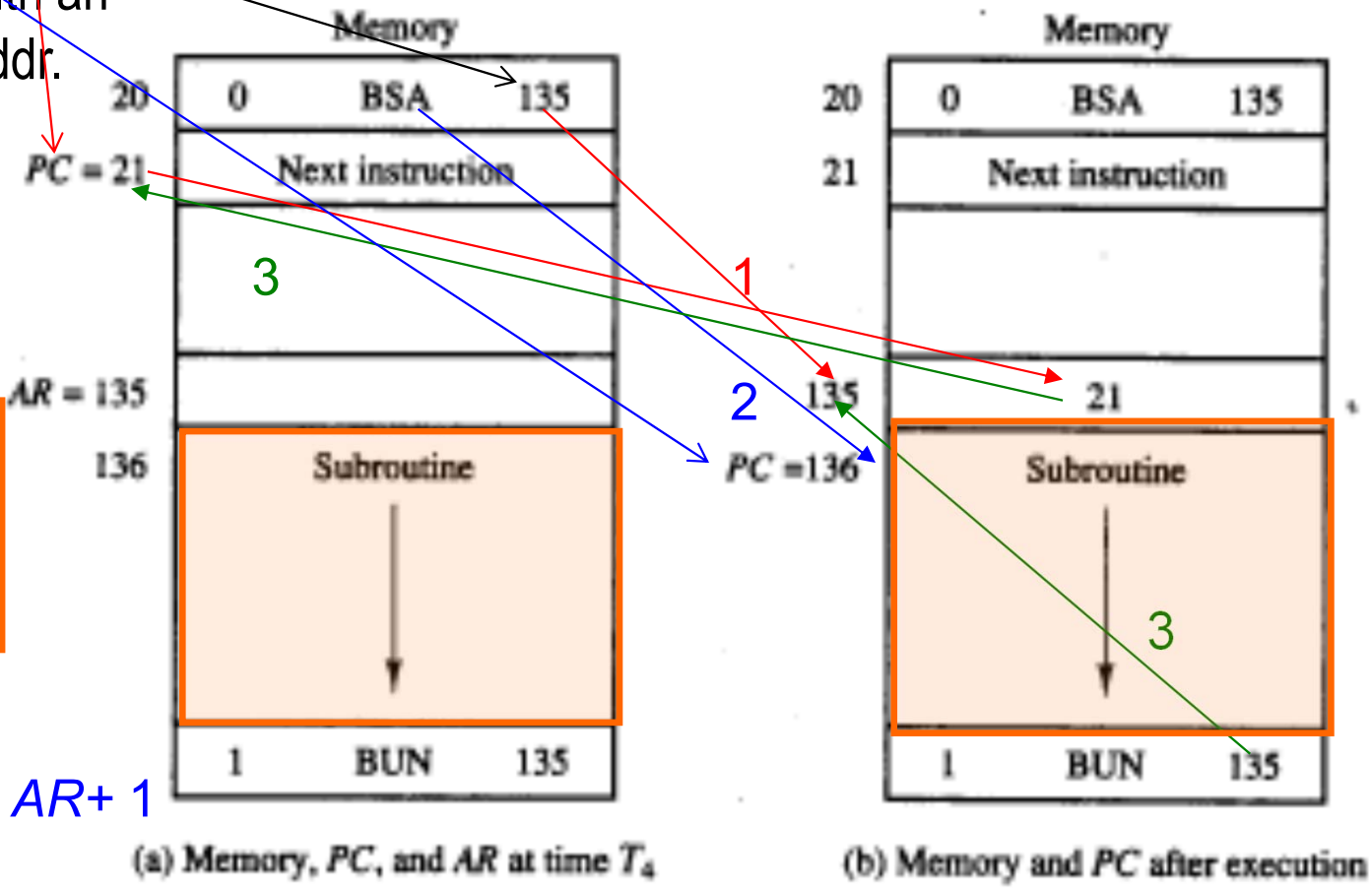
BSA: Branch and Save Return Address

The BSA instruction

1. ***stores the return address*** into the memory location specified by its effective address (135). *The* ***return address*** = the address of the instruction to be run after the subroutine is executed = 21 in our example; this address is already prepared in PC;
2. **branches to the first instruction of the *subroutine*** which is stored in the memory at the next address (136) after the stored return address.
3. The subroutine has to end with an **indirect BUN** to the return addr.

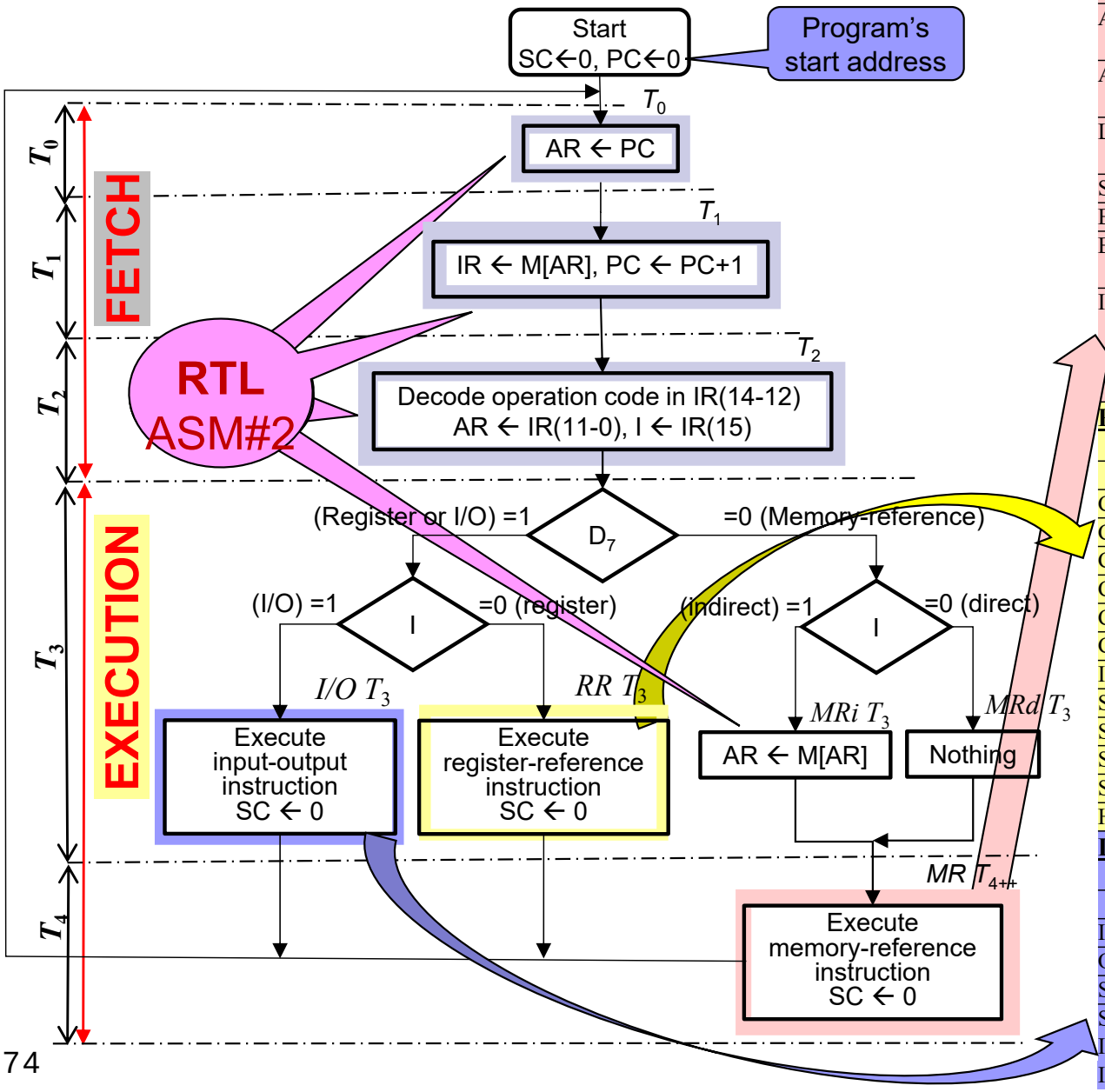
20	BSA 135
21	next instruction
22	……
….	…….
135	stored ret. addr (21)
136	Subroutine 1st instruc
137	…..
….	…..
159	Subroutine last instr
160	$BUN_{indirect}$ 135

1) D_5T_4: M[AR] ← PC, *AR* ← *AR*+ 1
2) D_5T_5: *PC* ← *AR*, SC ← 0



(a) Memory, *PC*, and *AR* at time T_4 (b) Memory and *PC* after execution

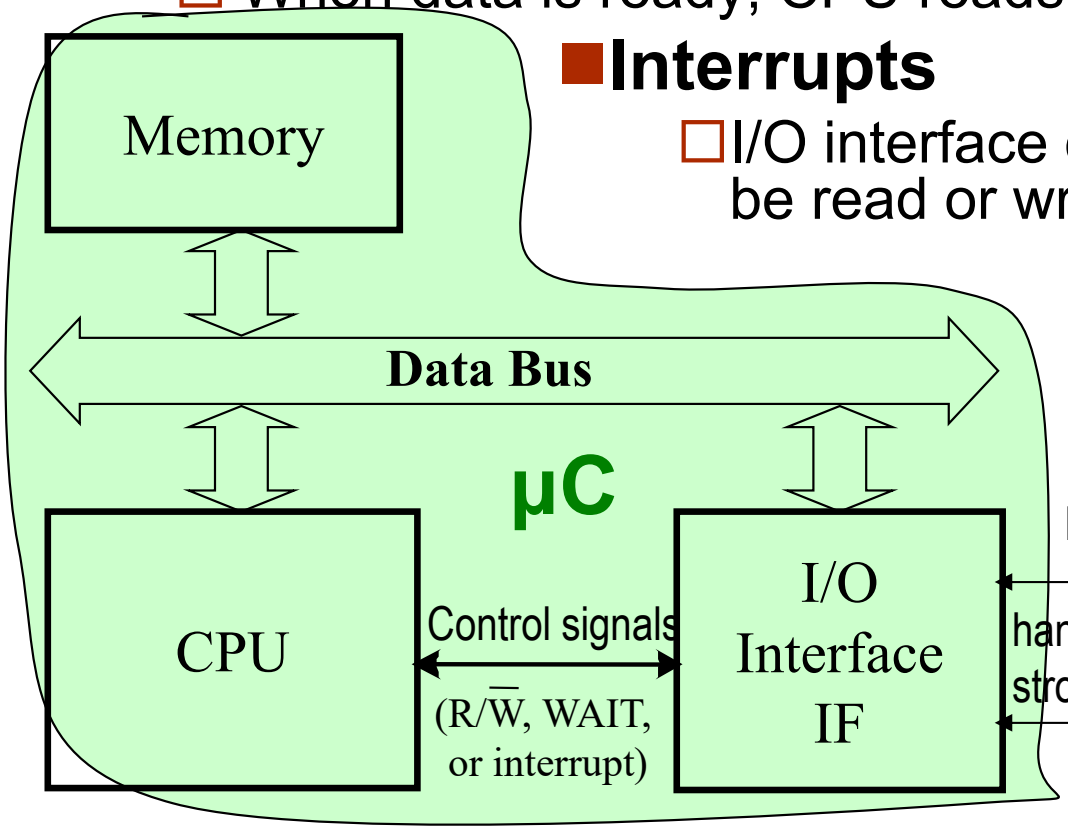
Control Functions & Microoperations of the Basic Computer



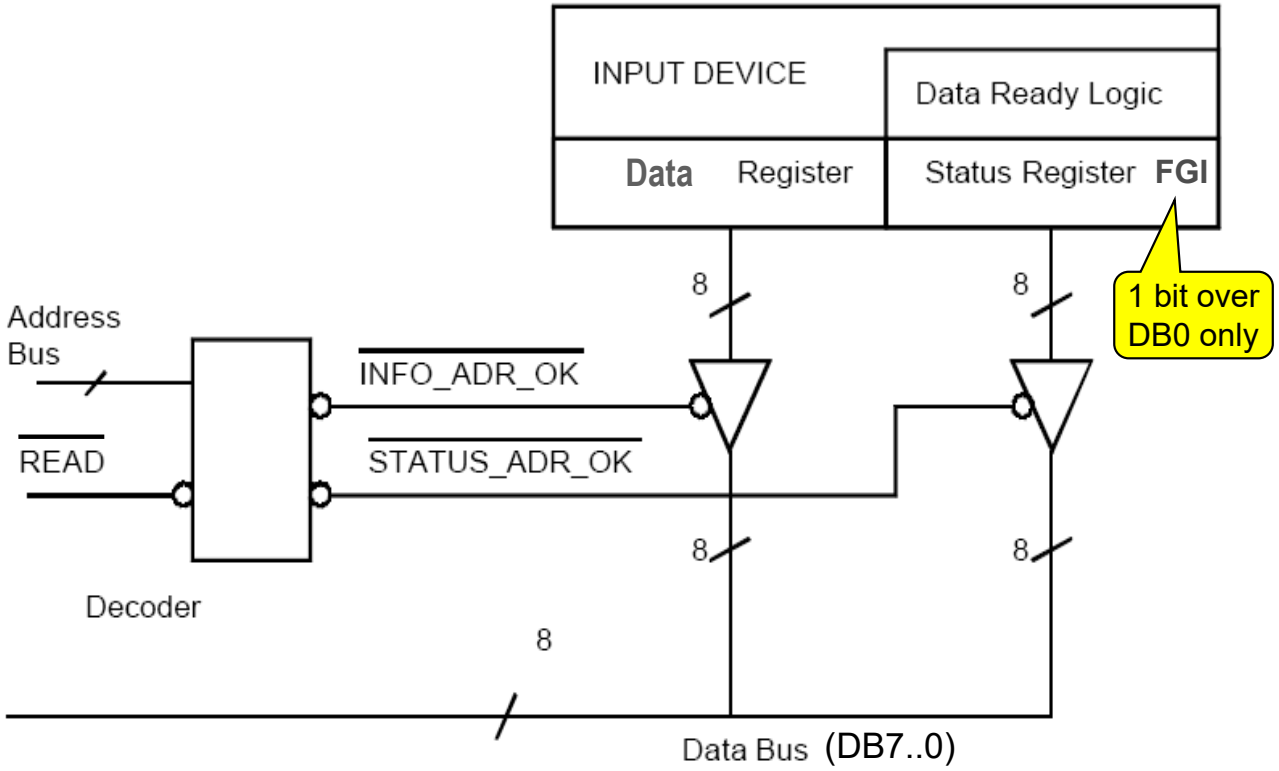
Fetch	T_0:	$AR \leftarrow PC$
	T_1:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	T_2:	$D_0, \ldots, D_7 \leftarrow Decode\ IR(12\text{-}14), AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Memory-reference:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$
STA.	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1), SC \leftarrow 0$
Reg-ref	$D_7I'T_3 = r$	(common to all reg.-ref. instr.) $IR(i) = B_i$ (i = 0, 1, 2, ... , 11)
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shl AC, AC(0) \leftarrow E, E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	$rB_4AC'(15)$:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	$rB_3AC(15)$:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2AC':	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$
SZE	rB_1E':	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
In-out		
INP		
OUT		
SKI		
SKO		
ION		
IOF		

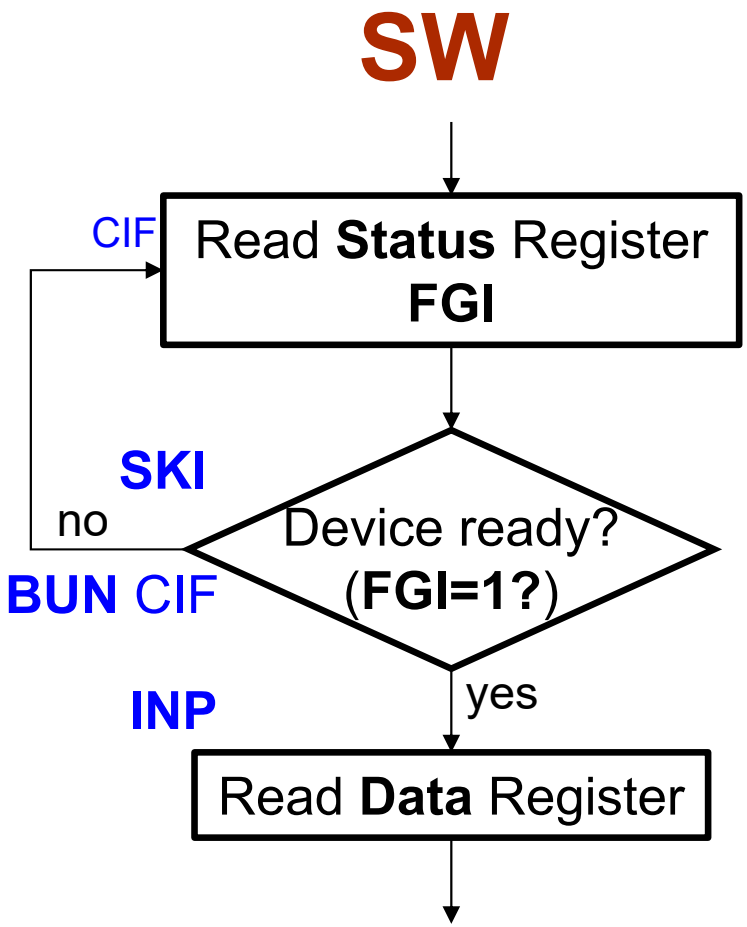
CPU & I/O Interface

- Hardware provides an **interface** to the I/O device. Contains:
  - data register (for data transfer, here: INPR and OUTR)
  - status register (flags=peripheral ready, here: FGI & FGO)
  - control register (here: R, IEN)
- **SW Polling** (Programmed Control)
  - CPU monitors the status register
  - When data is ready, CPU reads or writes from/to the data register
- **Interrupts**
  - I/O interface can interrupt the CPU when data can be read or written (will study later)
- **DMA (direct-memory access):**
  - the IF takes over the control of the system's Data, Address & Control Buses and transfers sequences of data direct to the memory.



Software Polling





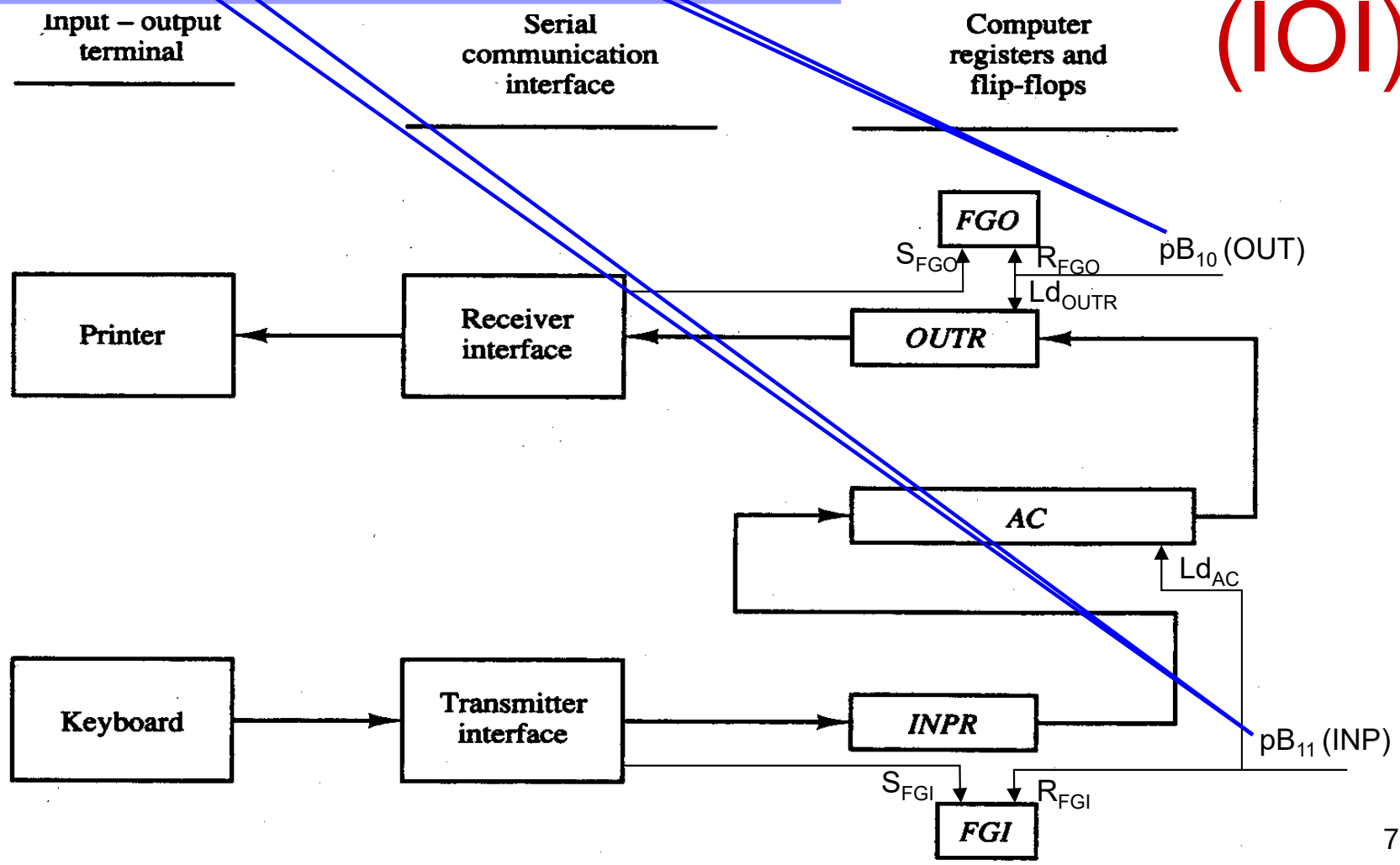
Basic Computer Input/Output Instructions (IOI)

$D_7IT_3 = p$ (common to all input–output instructions)
$IR(i) = B_i$ [bit in $IR(6–11)$ that specifies the instruction]

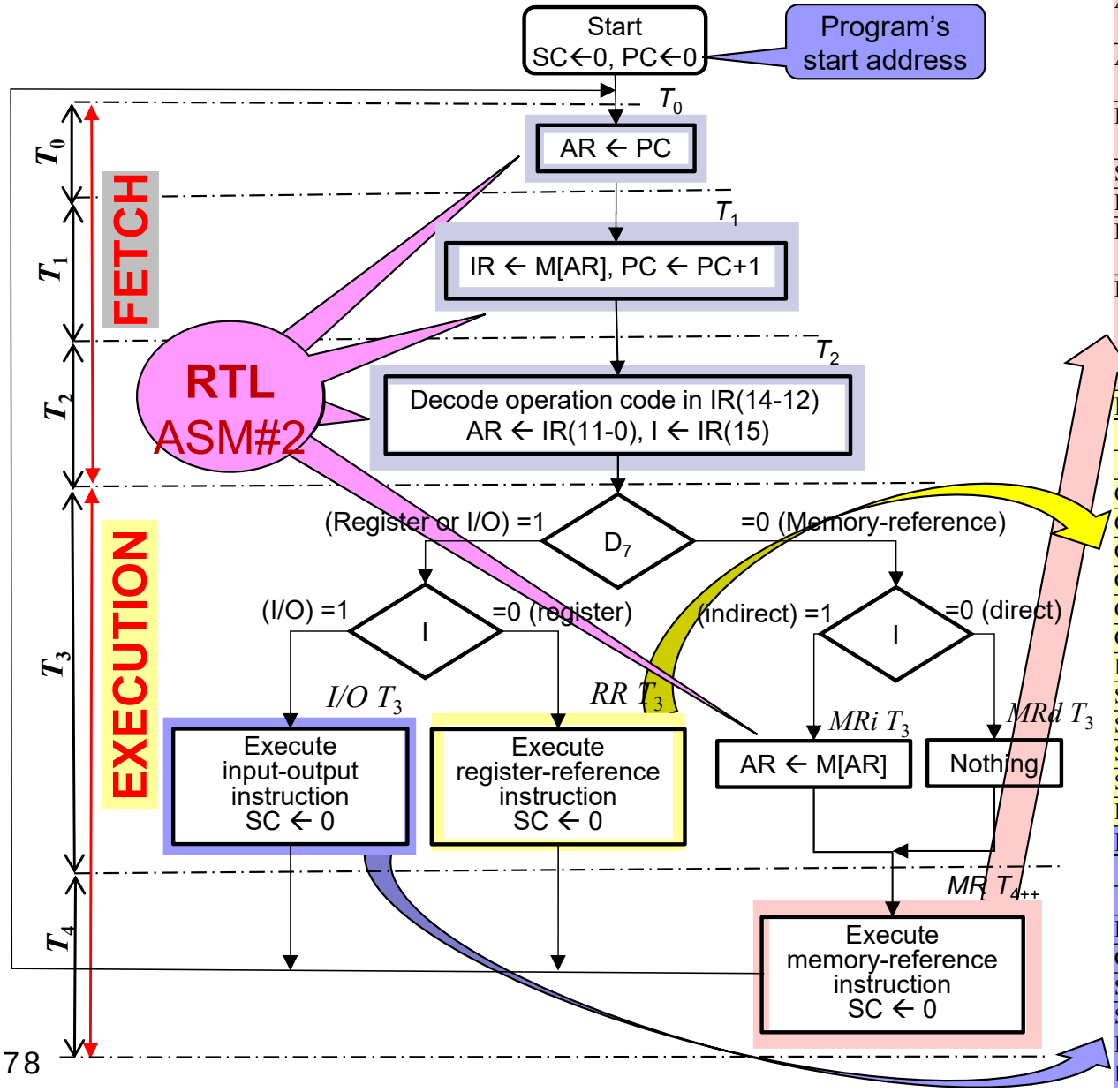
	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0–7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0–7)$, $FGO \leftarrow 0$	Output character
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

peripheral

RTL

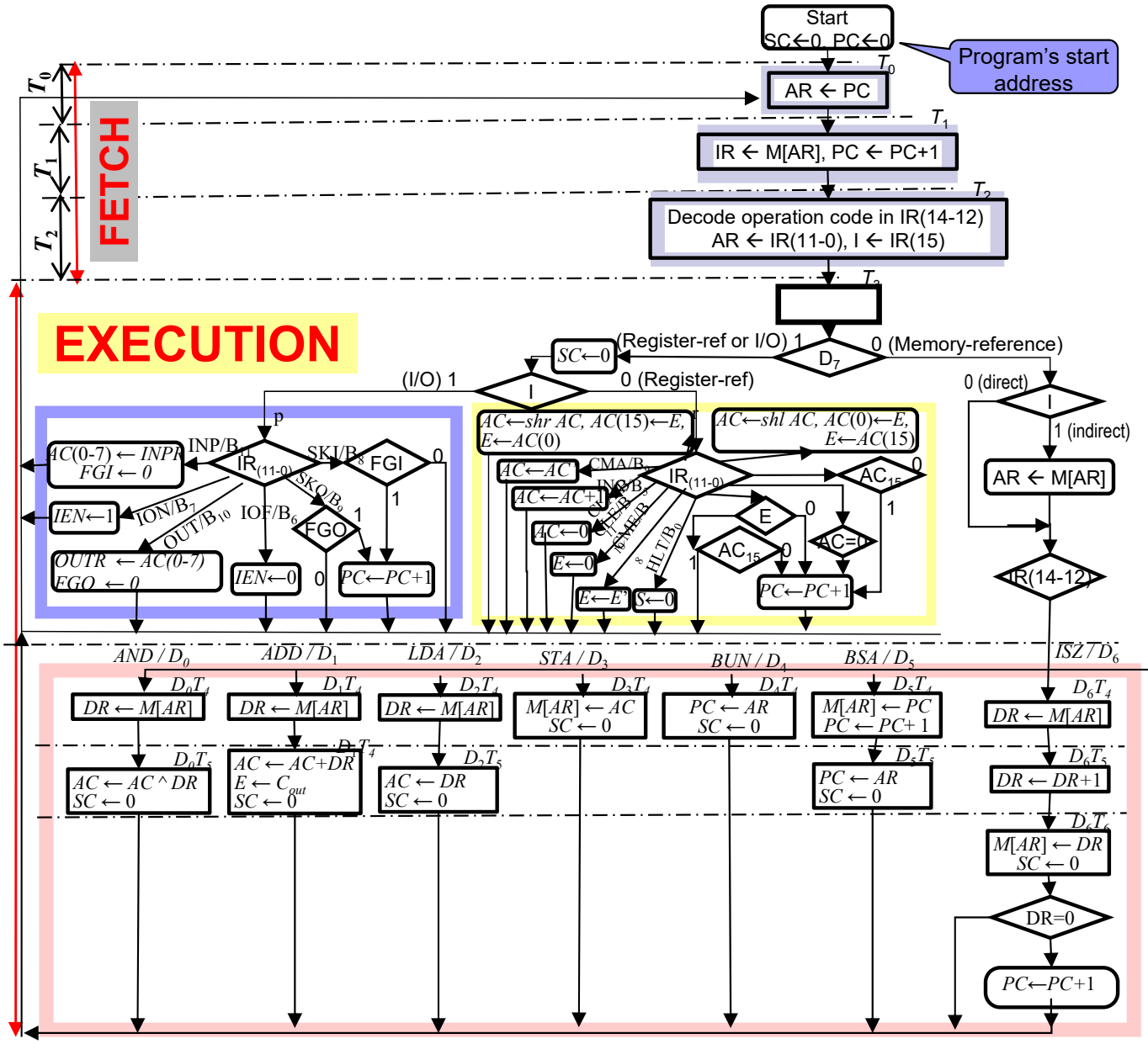


Control Functions & Microoperations of the Basic Computer



Fetch	T_0:	$AR \leftarrow PC$
	T_1:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	T_2:	$D_0, \ldots, D_7 \leftarrow$ Decode IR(12-14), $AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Memory-reference:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$
STA.	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1), SC \leftarrow 0$
Reg-ref	$D_7I'T_3 = r$	(common to all reg.-ref. instr.)
		$IR(i) = B_i$ (i = 0, 1, 2, ... , 11)
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shl AC, AC(0) \leftarrow E, E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	$rB_4AC'(15)$:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	$rB_3AC(15)$:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2AC':	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$
SZE	rB_1E':	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
In-out	$D_7IT_3 = p$	(common to all input-output instructions)
	$IR(i) = B_i$	(i = 6, 7, 8, 9, 10, 11)
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$
SKI	pB_9FGI:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8FGO:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

Control Functions & Microoperations of the Basic Computer (ASM RTL level)



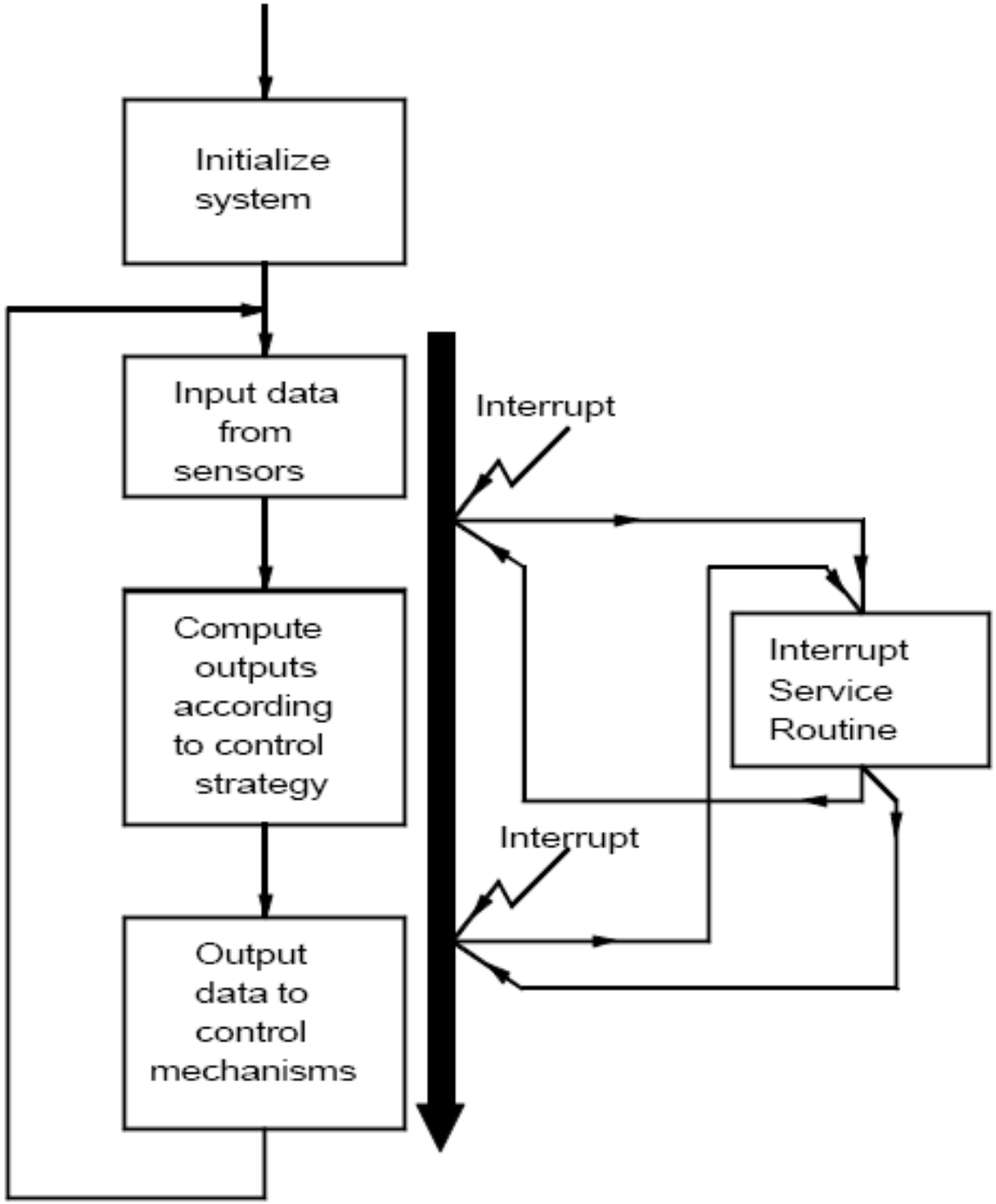
Fetch	T_0:	$AR \leftarrow PC$
	T_1:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	T_2:	$D_0, \ldots, D_7 \leftarrow Decode\ IR(12\text{-}14), AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$
Indirect	D'_7IT_3	$AR \leftarrow M[AR]$
	:	

MemoryRef.		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$
STA.	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if $(DR=0)$ then $(PC \leftarrow PC+1)$, $SC \leftarrow 0$

Reg-ref	$D_7I'T_3 = r$ (common to all reg.-ref. instr.) $IR(i) = B_i$ (i = 0, 1, 2, ... , 11)	
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shl AC, AC(0) \leftarrow E, E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2:	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$

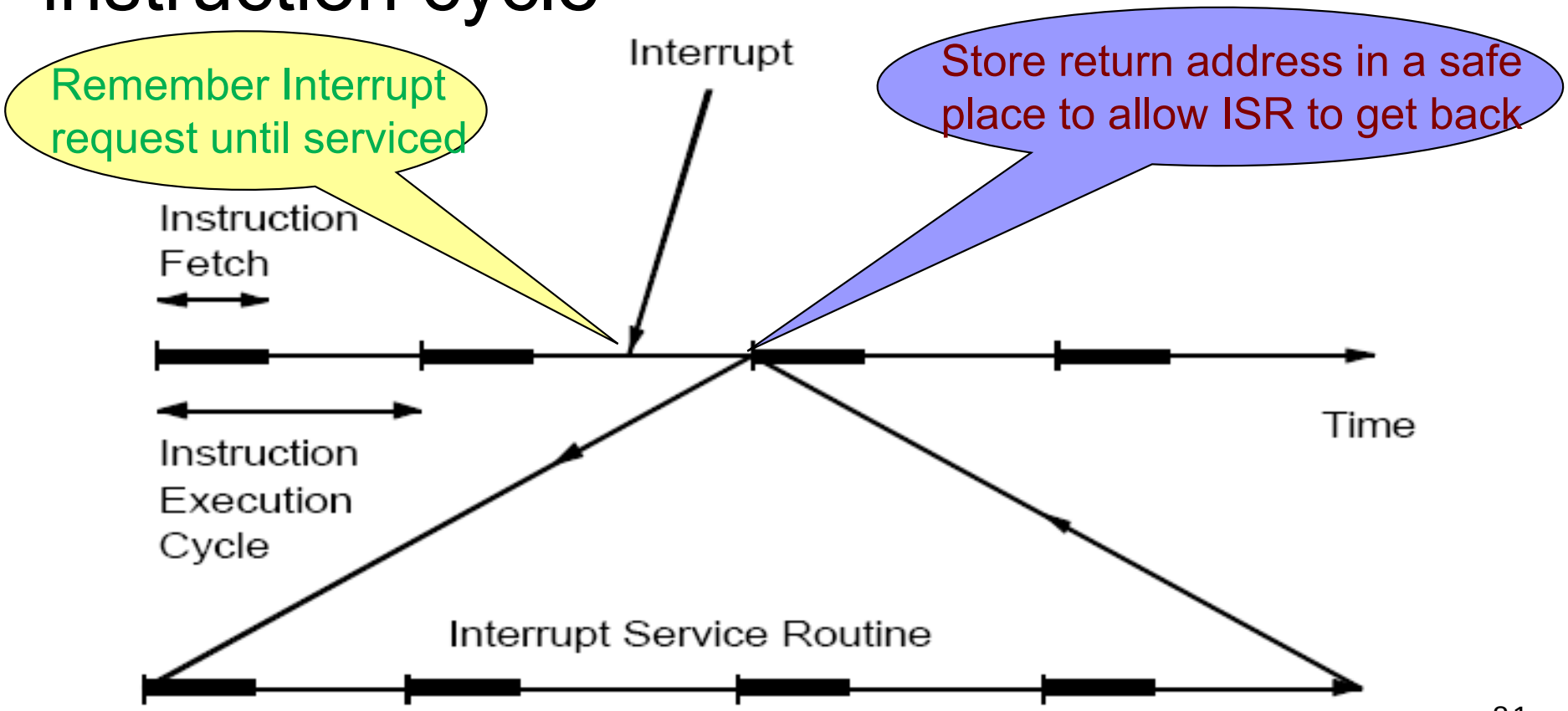
In-out	$D_7IT_3 = p$ (common to all I/O instructions) $IR(i) = B_i$ (i = 6, 7, 8, 9, 10, 11)	
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

Interrupt Fundamental Concepts



Internal Asynchronous CPU Timing

- Interrupts can occur at any time during an instruction cycle

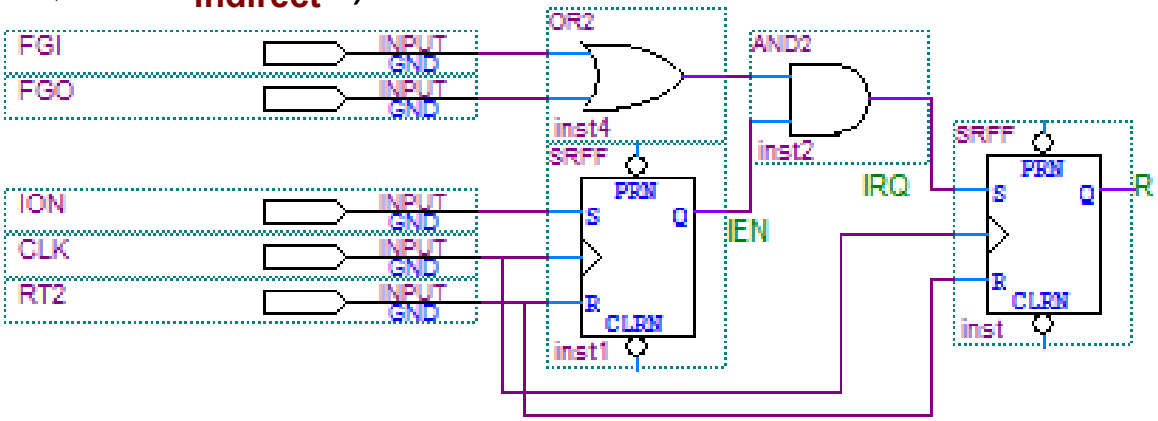


Interrupt System Specification

- Allow for **asynchronous events** to occur and be recognized.
- Wait for the current instruction to finish before servicing an interrupt.
- Service interrupt with a sub-routine (Interrupt Service Routine = ISR) and returns to interrupted code.
- Enabling and disabling interrupts (IEN)
- Multiple sources of interrupts (here 2: FGO, FGI)
  - Simultaneous interrupts.

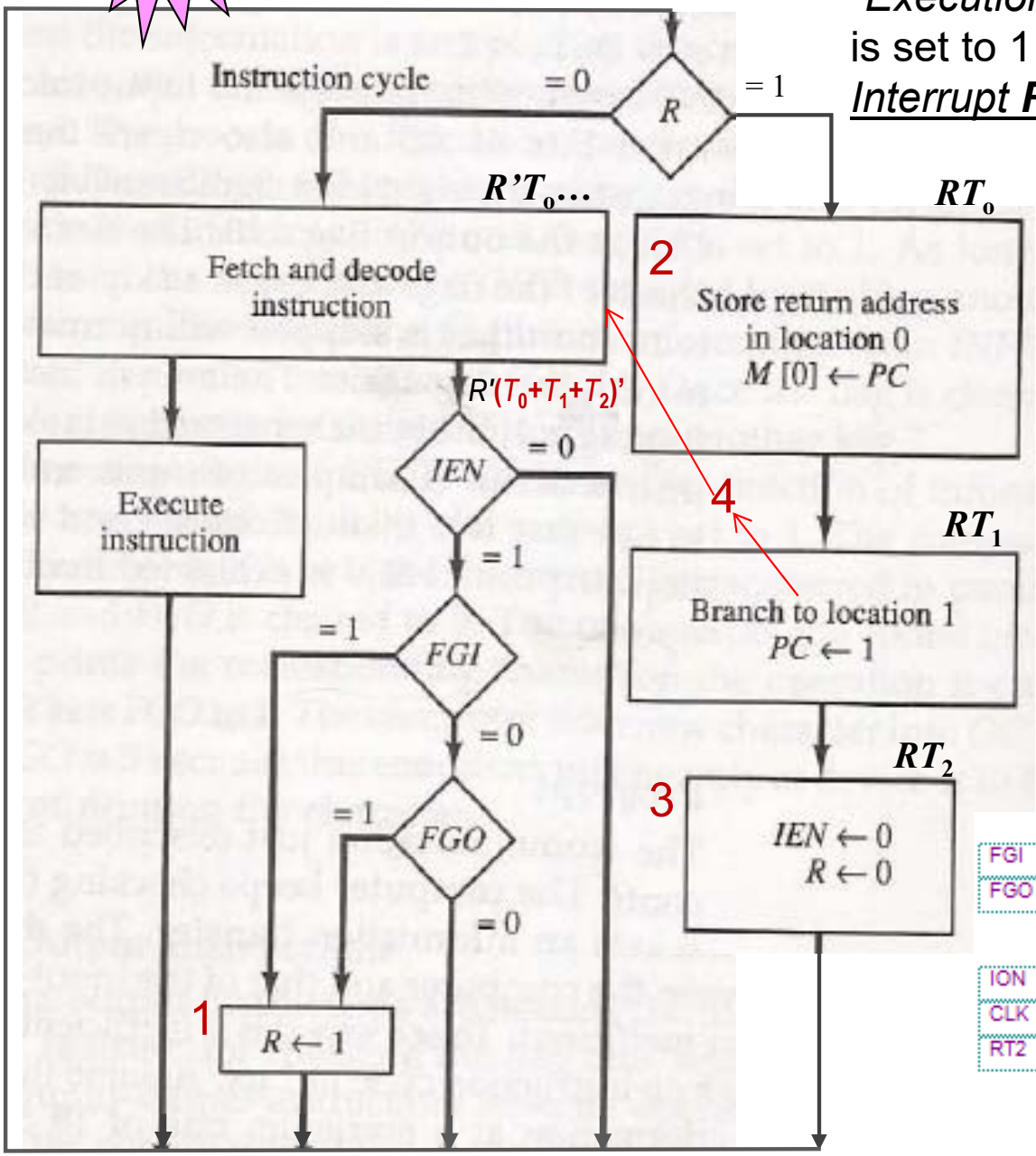
Internal CPU Interrupt Hardware

- Use a flip-flop (R) to catch IRQ (FGI, FGO)
  - Multiple device interrupt lines (FGI, FGO) can be OR-wired together
  - CPU waits until the current instruction is executed and then can process interrupt
- Can enable/disable interrupt using enable-disable flip-flop (IEN)
  - When interrupt is acknowledged by CPU HW, interrupts are disabled (IEN ← 0) for the duration of interrupt servicing
- The source of interrupt is determined by FGI or FGO → the Interrupt Service Routine (ISR) has to check which I/O device generated the IRQ
- The return address is stored at address 0; at the end of ISR
  - ION is executed to arm the interrupt system IEN ← 1
  - **BUN 0 I** (i.e., **BUN$_{indirect}$0**) is executed to return to the interrupted program



Interrupt Cycle

RTL



1. Once the "*Fetch*" phase is finished, during the "*Execution*" phase $(T_0+T_1+T_2)'$, if a flag (***FGI*** or ***FGO***) is set to 1 by an interrupt, the computer stores the *Interrupt* ***Request*** in ***R*** if enabled by ***IEN*** :

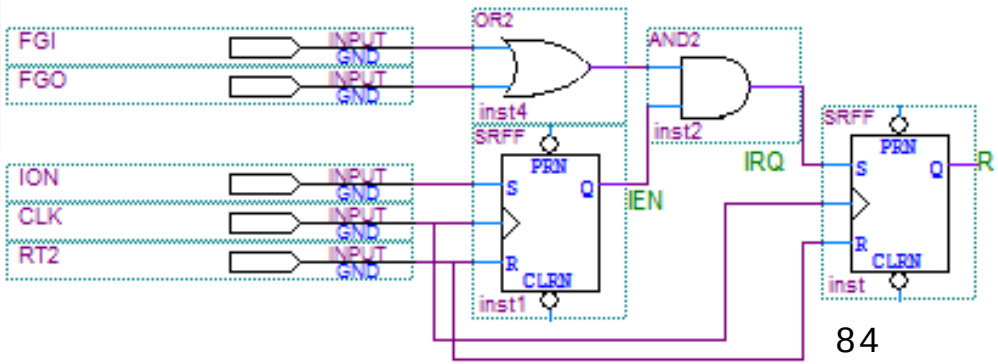
$(T_0 + T_1 + T_2)'(IEN)\ (FGI + FGO)$: $R \leftarrow 1$

and completes the execution of the instruction in progress;

2. Next goes to an Interrupt cycle, where stores the return address in the location 0 of the memory;

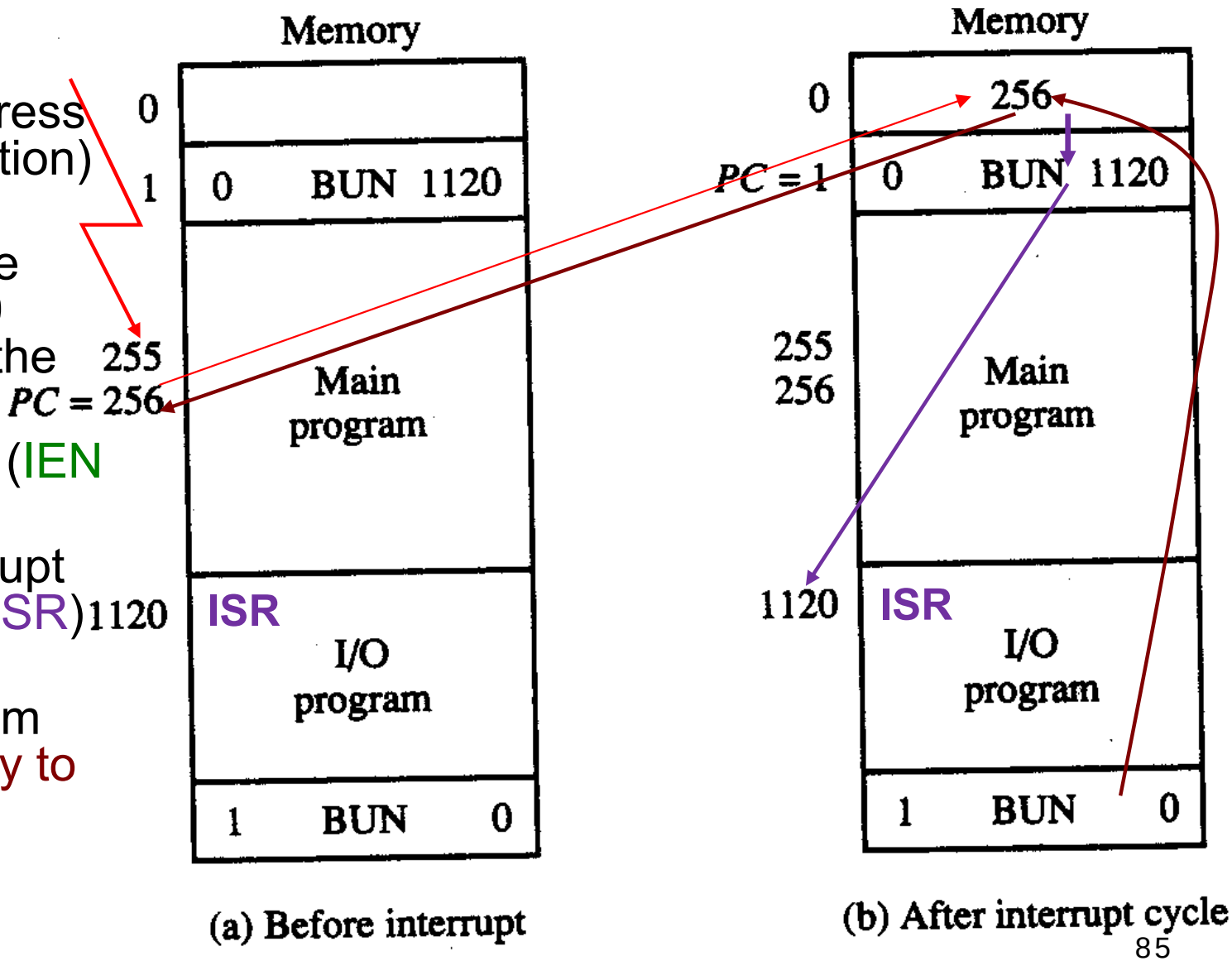
3. Then disables the interrupt system (***IEN***=0) and resets *R* (since the system already takes care of this interrupt);

4. Executes the instruction stored in the memory location 1, which is a direct unconditional branching to the base address of the service routine.

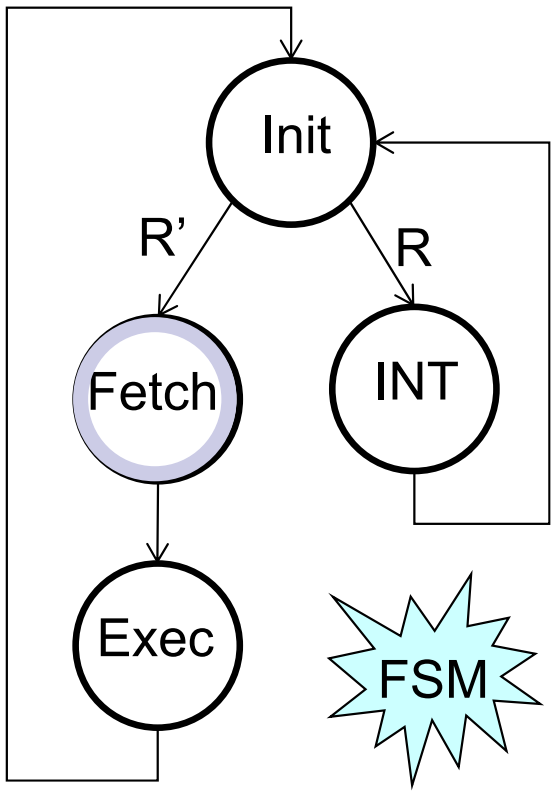


Interrupt Service Cycle (INT)

1. Save the return address (PC=address of the next instruction) at the address 0;
2. BUN directly to the ISR address 1120 corresponding to the interrupt
3. Disable interrupts (IEN ← 0, R ← 0)
4. Execute the Interrupt Service Routine (ISR)
5. Resume the interrupted program with BUN indirectly to address 0



Basic Computer Operation



RTL

Start
$SC \leftarrow 0, IEN \leftarrow 0, R \leftarrow 0$

R
(instruction cycle) = 0
= 1 (interrupt cycle)

$R'T_0$: $AR \leftarrow PC$

$R'T_1$: $IR \leftarrow M[AR], PC \leftarrow PC + 1$

$R'T_2$: $AR \leftarrow IR(0-11), I \leftarrow IR(15)$
$D_0 \cdots D_7 \leftarrow$ Decode $IR(12-14)$

RT_0: $AR \leftarrow 0, TR \leftarrow PC$

RT_1: $M[AR] \leftarrow TR, PC \leftarrow 0$

RT_2: $PC \leftarrow PC + 1, IEN \leftarrow 0$
$R \leftarrow 0, SC \leftarrow 0$

D_7
(Register or I/0) = 1
= 0 (Memory – reference)

I
(I/0) = 1
= 0 (register)

I
(indirect) = 1
= 0 (direct)

$D_7 I T_3$: Execute input-output instruction (Table 5-5)

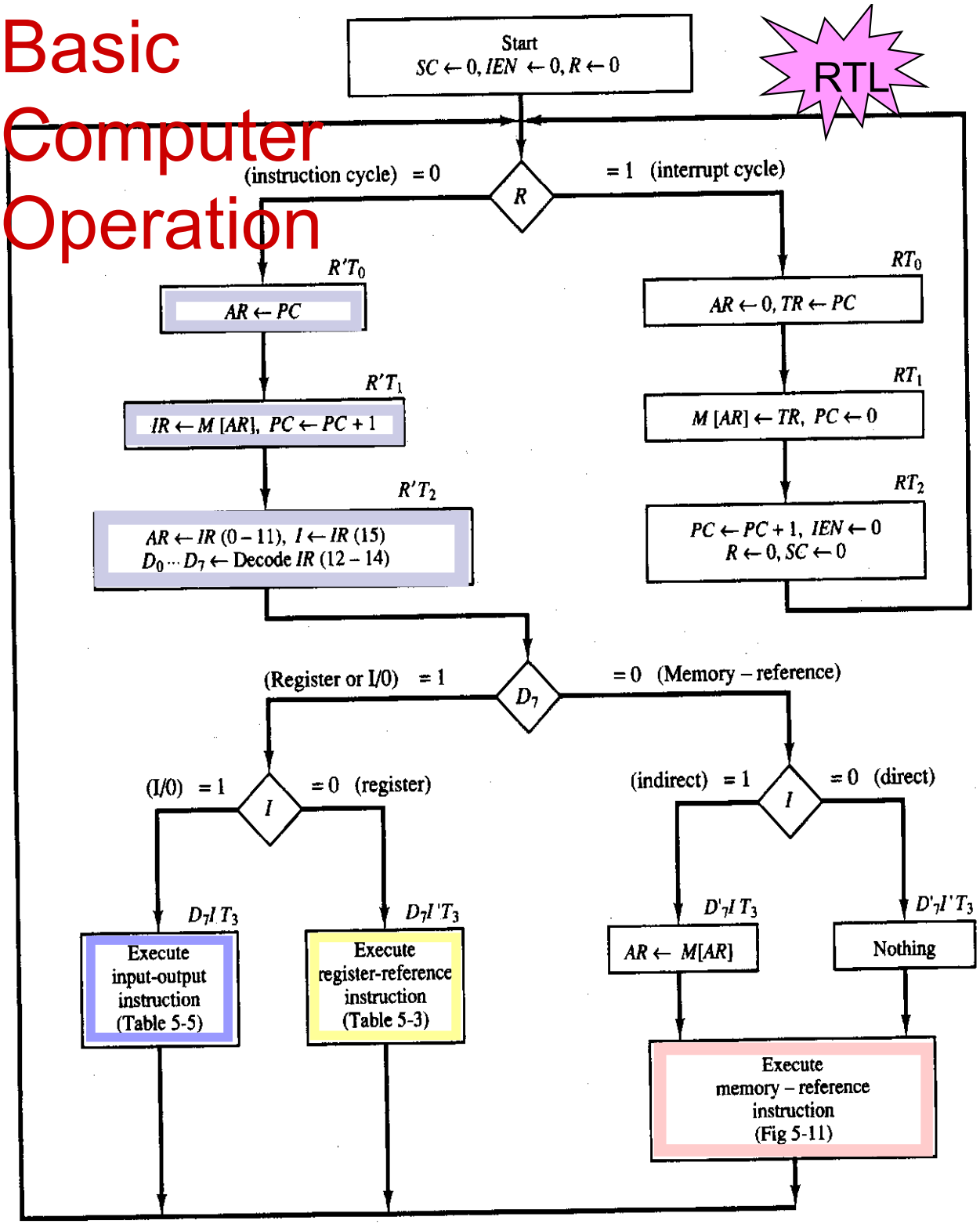
$D_7 I' T_3$: Execute register-reference instruction (Table 5-3)

$D'_7 I T_3$: $AR \leftarrow M[AR]$

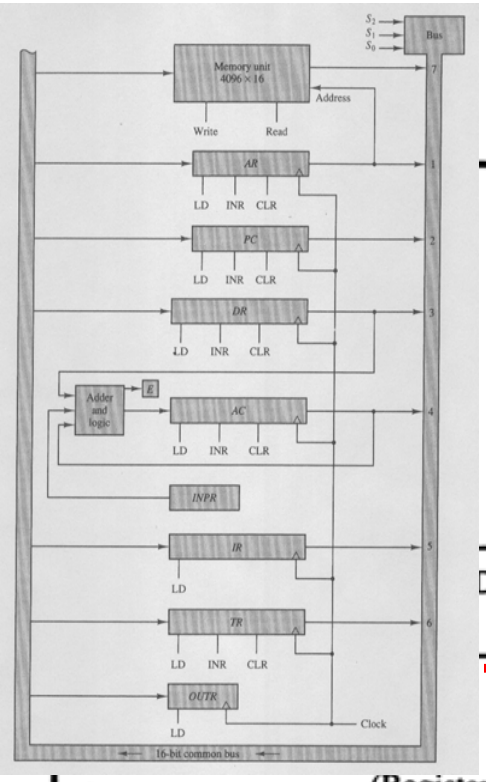
$D'_7 I' T_3$: Nothing

Execute memory – reference instruction (Fig 5-11)

Basic Computer Operation



Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{-}14)$, $AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Interrupt:	$\overline{T_0}\,\overline{T_1}\,\overline{T_2}(IEN)(FGI + FGO)$:	$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC+1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$
Memory-reference:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$
STA.	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1), SC \leftarrow 0$
Reg-ref	$D_7I'T_3 = r$	(common to all reg.-ref. instr.)
	$IR(i) = B_i$ (i = 0, 1, 2, ... , 11)	
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shlAC, AC(0) \leftarrow E, E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4AC'(15):	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	rB_3AC(15):	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2AC':	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$
SZE	rB_1E':	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
In-out	$D_7IT_3 = p$	(common to all input-output instructions)
	$IR(i) = B_i$	(i = 6, 7, 8, 9, 10, 11)
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$
SKI	pB_9FGI:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8FGO:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$



Instruction Cycle

Ottawa

Start
SC ← 0

T_0: AR ← PC

Fetch instruction (opcode + op addr)

T_1: IR ← M [AR], PC ← PC + 1

T_2: Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)

T_3

D_7: (Register or I/0) = 1 / = 0 (Memory-reference)

I: (I/0) = 1 / = 0 (register)

I: (indirect) = 1 / = 0 (direct)

operand addr, if indirect addressing

Execute input-output instruction
SC ← 0

Execute register-reference instruction
SC ← 0

AR ← M[AR]

Nothing

Execute memory-reference instruction
SC ← 0

ASM#2 Flowchart

Control Signals

RTL

RRI

IOI

FETCH

EXECUTION

MRI

T_0: $S_2'S_1S_0'$: LD_{AR}, INR_{SC}

$D_7'I'(STA)/S_2S_1'S_0'$: COM, CLR_{SC}

$D_7'I'(INC)/S_2S_1'S_0'$: INR_{PC}, CLR_{SC}

$D_7I(INP)/S_2'S_1'S_0'$: LD_{AC}, CLR_{SC}

T_1: $S_2S_1S_0'$: LD_{IR}, INR_{PC}, Read, INR_{SC}

Fetch instruction

T_2: $S_2S_1'S_0'$: LD_{AR}, INR_{SC}

T_3

$(STA)/S_2S_1'S_0'$: Write, CLR_{SC}

$(ISZ)/S_2'S_1S_0'$: DR', INR_{PC}, CLR_{SC}, Write

$(ADD)/ADD, S_2'S_1'S_0'$: CLR_{SC}

T_4: $D_7'I'/Ø$ V $D_7'I/S_2S_1S_0$, LD(AR), Read INR_{SC}

operand address, if indirect addressing

$(ADD)/LD\ DR$: $S_2S_1S_0$, Read, INR_{SC}

operand

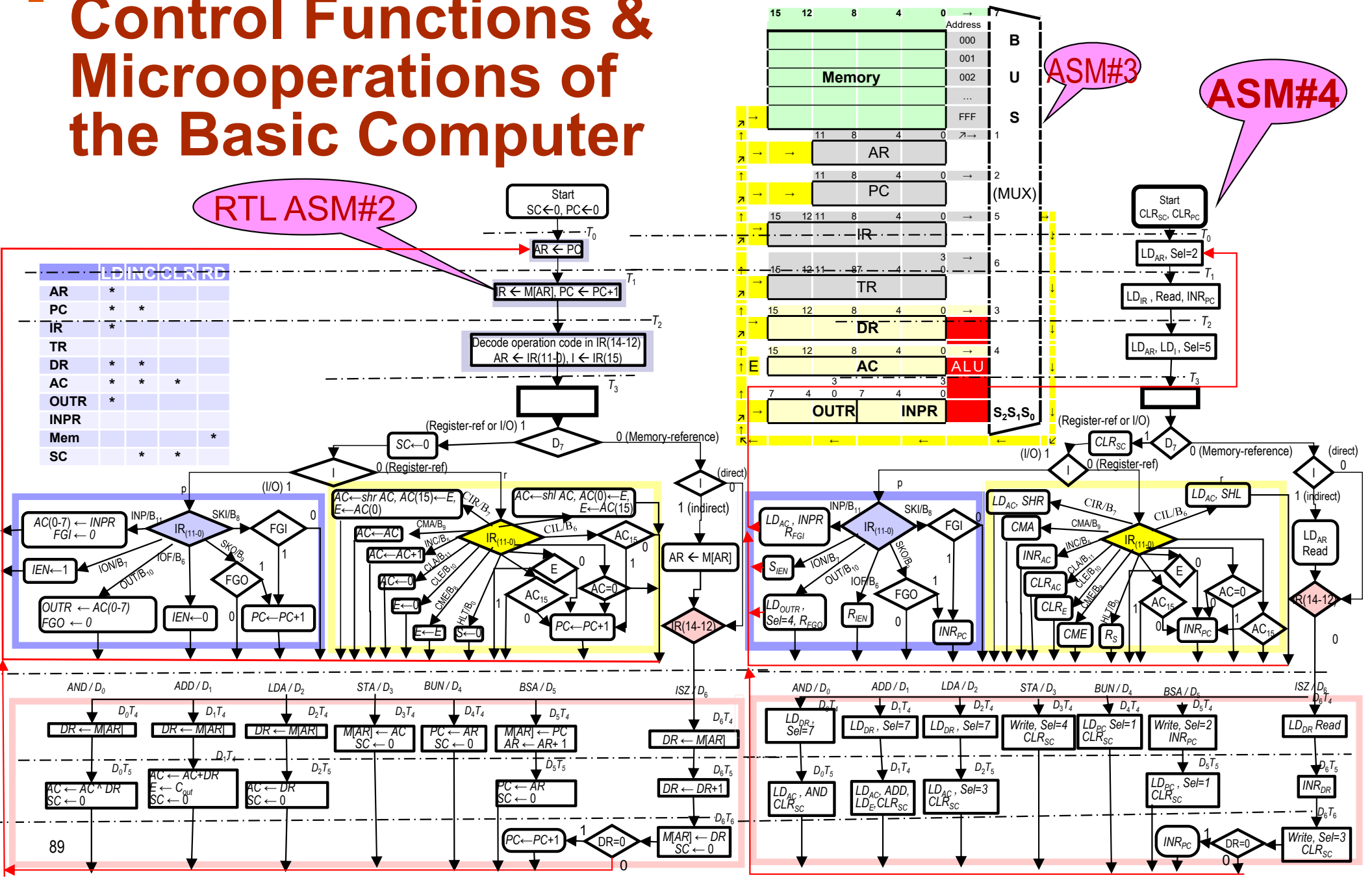
T_5

$(ISZ)/S_2'S_1'S_0'$: INR_{DR}, INR_{SC}

T_6

FSM ASM#4

Control Functions & Microoperations of the Basic Computer



Control Unit Design

Inputs: X = {D_i, Other inputs}
IR => D_i Instruction Decoder - AND, ADD
LDA, STA, BUN, BSA, ISZ, {RRI, IOI}

Output: Z = {LD_{AR}, INR_{AR}, CLR_{AR},
LD_{PC}, INR_{PC}, CLR_{PC},
Write, Read, LD_{IR},
LD_{DR}, INR_{DR}, CLR_{DR},
LD_{TR}, INR_{TR}, CLR_{TR},
LD_{AC}, INR_{AC}, …}

States : {T_0, T_1, T_2, T_3, … , T_{15}}

$T^{n+1} = \delta(X^n, T^n)$ next state function

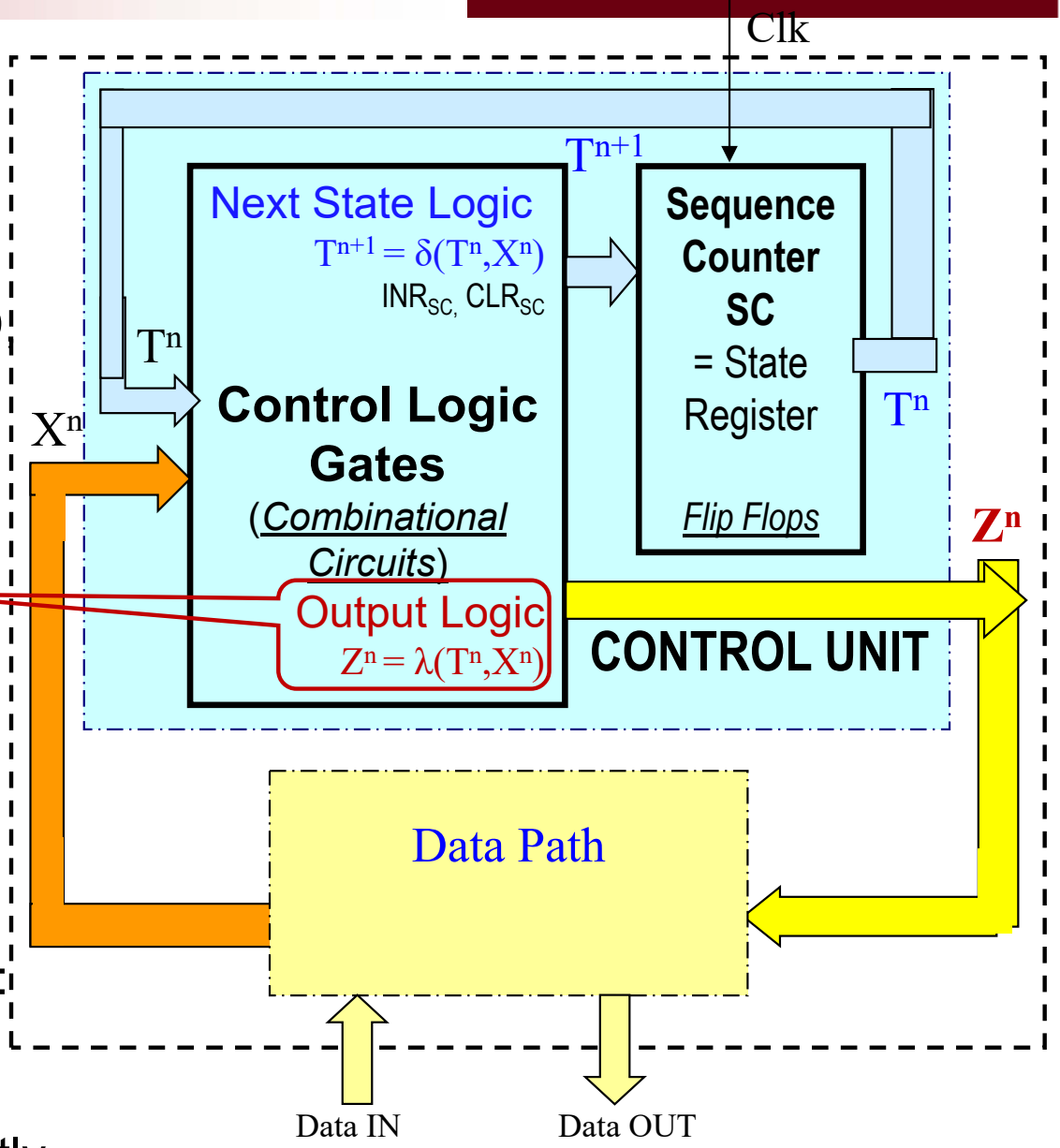
$Z^n = \lambda(X^n, T^n)$ output function

To find the equations for λ & δ one can:

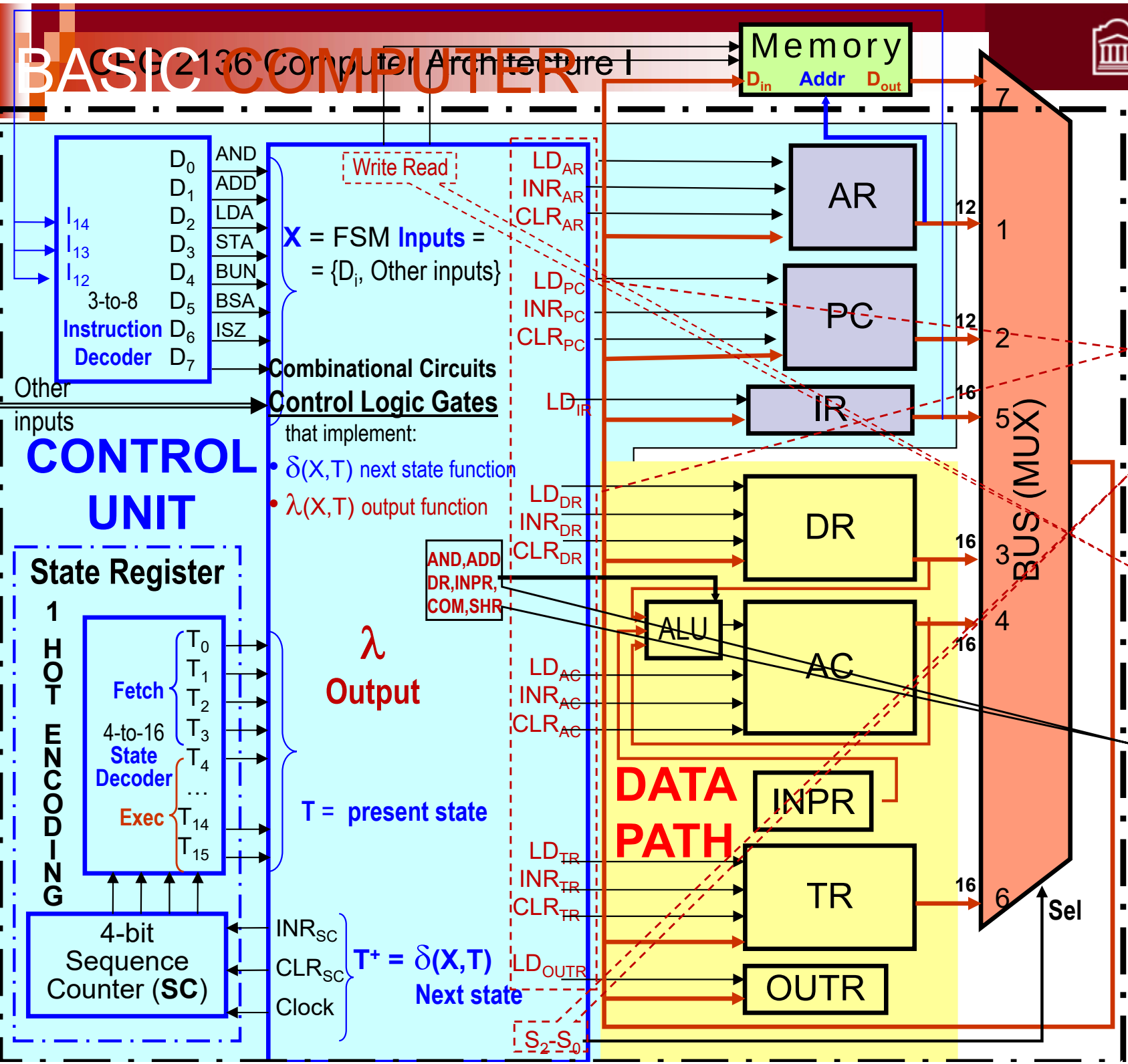
a. Scan Table 5-6 & find all the RTL statements in which registers are modified and derive equations directly

OR:

b. or c. Extend Table 5.6 with columns for each set of signals needed to perform every RTL and derive signals' equations from these columns.



BASIC COMPUTER



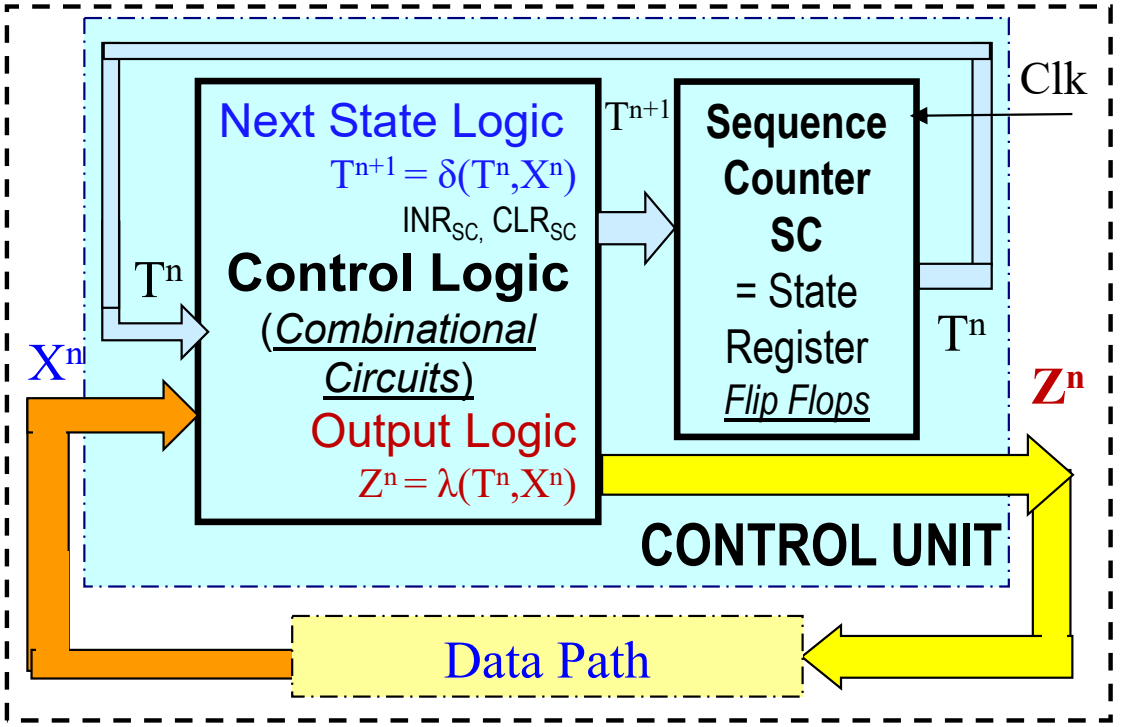
Control Unit

The outputs λ of the control logic circuit in the control unit are:

1. Signals to control the inputs (LD, INR, and CLR) of the **8 registers +SC** (CLR)
2. Signals to control common **bus** selection bits: S2, S1, and S0.
3. Signals to control the Read and Write inputs of **memory**.
4. Signals to set, clear or complement individual **flip-flops**.
5. Signals to control the **AC** adder and logic circuit.

The specifications for the various control signals can be obtained directly from the list of register transfer statements in Table 5-6 of the textbook (page 159).

ASM#5 Synthesis of Control Unit



The major methods for implementing the State Register of sequential circuits which are used as **control logic** ASM-based circuits are:

1. The **One-hot Encoded State** method $\{T_0, T_1, T_2, T_3, \ldots, T_{15}\}$
   - Most popular design technique, as it is simple and easy to model
   - Implemented as
     - **one-FF-per-state**
     - **sequence-counter**
       - Matches the cyclical nature of instruction execution on a computer
       - Control is simplified because of the cyclical state transition pattern
2. The **Binary Encoded State** method
   - Uses state encoding to reduce the number of D flip-flops required in the state register
   - Control logic is spread out over several different digital devices

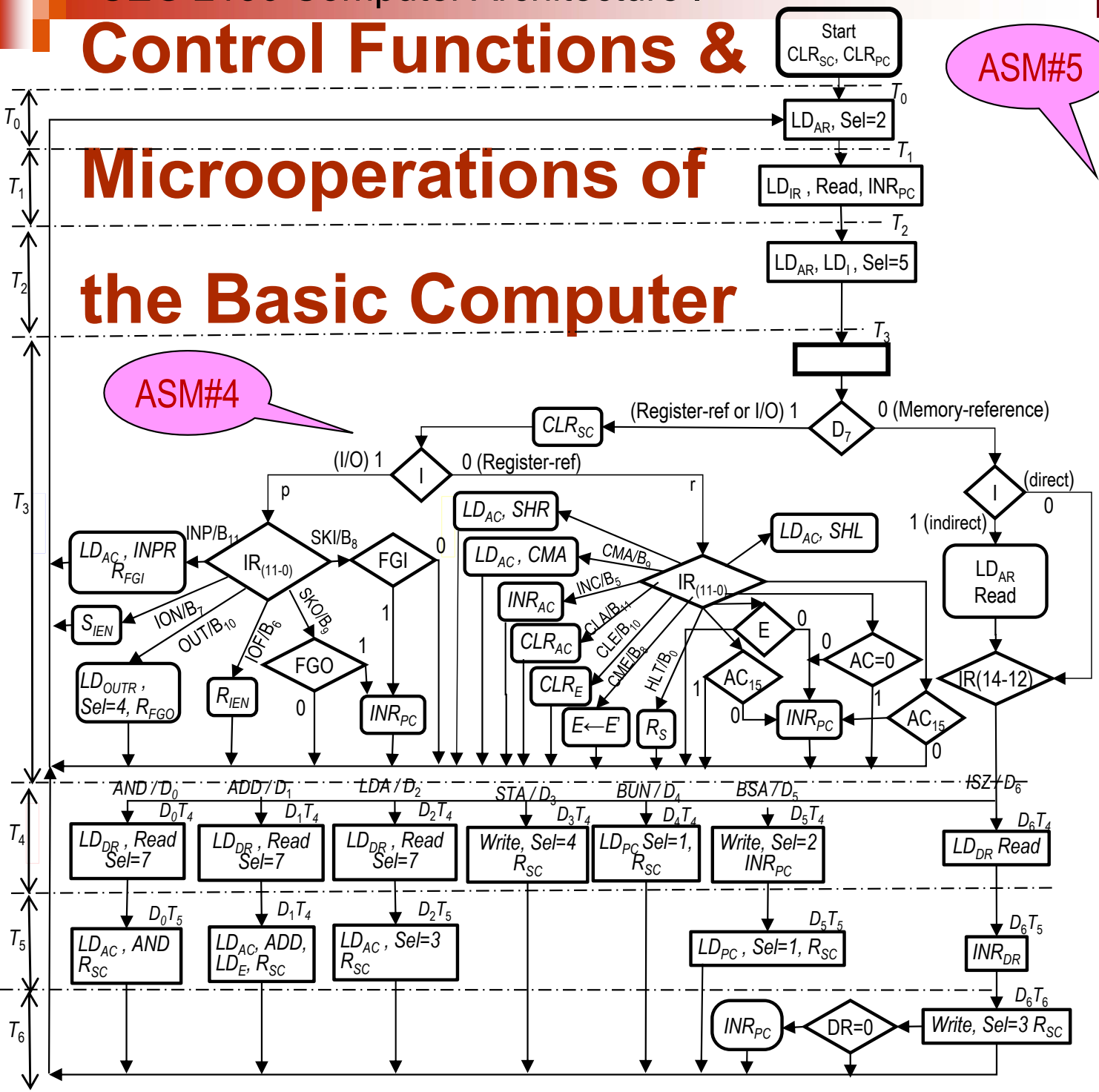
Control Units can be implemented as

- **Hardwired CU** - with gates and other MSI components (MUX, Decoders, multi-function registers) – Chapter 5

  Based on transition table derived from ASM#4
  a) Synthesis with gates (slide 117)
  b) Synthesis with MUX's (slide 123)
  c) Direct Synthesis with Gates & Decoder (124)
  d) Direct Synthesis with MUX's (125)
- **Microprogram CU** - ROM based.- Ch 7

uOttawa

Control Functions & Microoperations of the Basic Computer



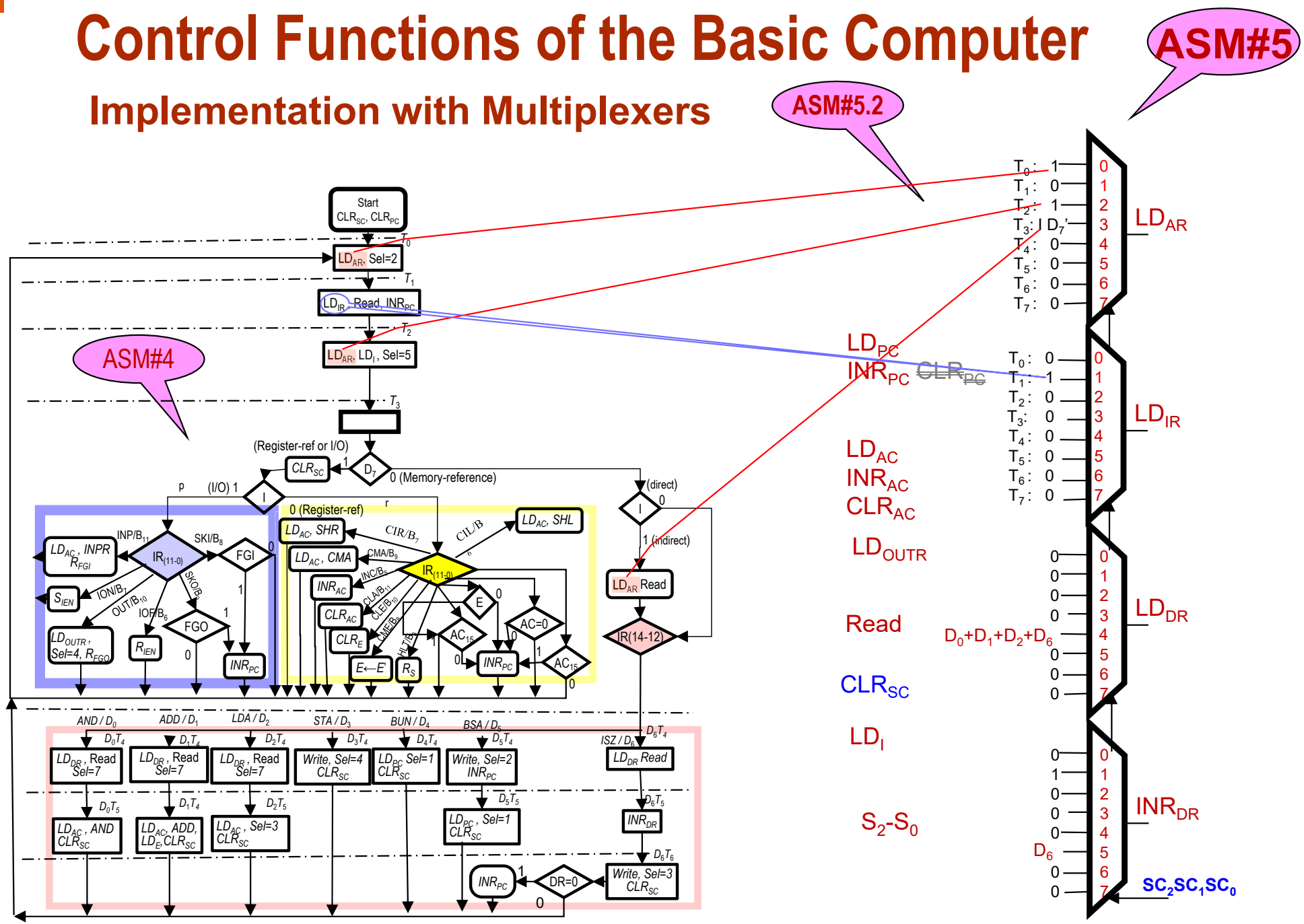
The outputs λ of the control logic circuit in the control unit are:

1. Signals to control the inputs (LD, INR, and CLR) of the **8 registers + SC** (CLR)
2. Signals to control common **bus** selection bits: S2, S1, and S0.
3. Signals to control the Read and Write inputs of **memory**.
4. Signals to set, clear or complement individual **flip-flops**.
5. Signals to control the **AC** adder and logic circuit.

The equations of the various control signals can be obtained directly from this ASM#4 state-chart.

Control Functions of the Basic Computer

Implementation with Multiplexers



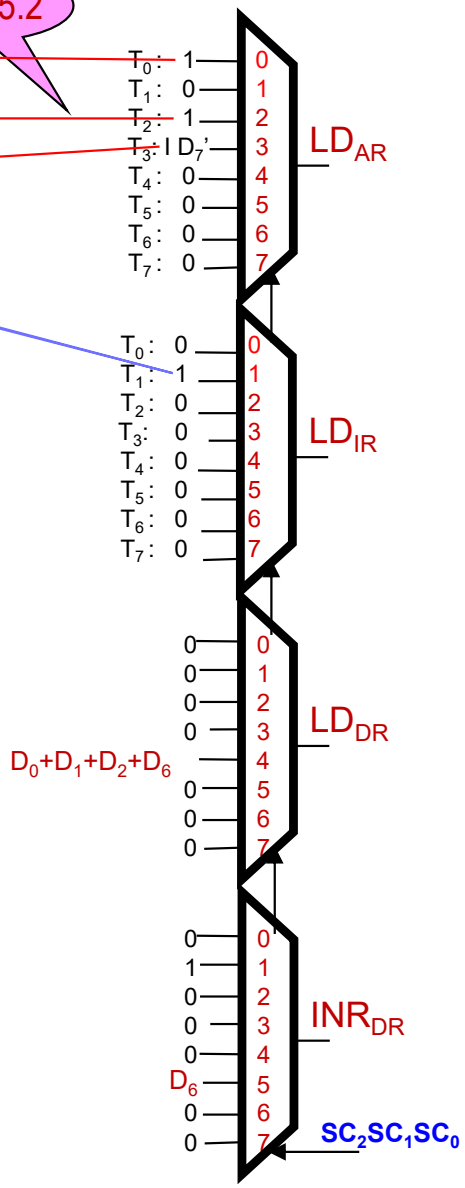
Control Functions of the Basic Computer

Implementation with Multiplexers

ASM#5

ASM#5.2

Fetch	T_0:	$AR \leftarrow PC$	LD_{AR}, Sel=2 ($S_2S_1S_0$=010)
	T_1:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$	LD_{IR} , Read, INR_{PC}
Decode	T_2:	$D_0, \ldots, D_7 \leftarrow$ Decode IR(12-14), $AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$	LD_{AR}, LD_I , Sel=5
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$	
Memory-Ref.			
AND	D_0T_4:	$DR \leftarrow M[AR]$	LD_{DR} , Read, Sel=7
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$	LD_{AC} , AND, CLR_{SC}
ADD	D_1T_4:	$DR \leftarrow M[AR]$	LD_{DR} , Read, Sel=7
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$	LD_{AC} , ADD, CLR_{SC}
LDA	D_2T_4:	$DR \leftarrow M[AR]$	LD_{DR} , Read, Sel=7
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$	LD_{AC} , Sel=3, CLR_{SC}
STA	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$	Write, Sel=4, CLR_{SC}
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$	LD_{PC} Sel=1, CLR_{SC}
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$	Write, Sel=2 INR_{AR}
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$	LD_{PC} , Sel=1, CLR_{SC}
ISZ	D_6T_4:	$DR \leftarrow M[AR]$	LD_{DR} , Read, Sel=7
	D_6T_5:	$DR \leftarrow DR + 1$	INR_{DR}
	D_6T_6:	$M[AR] \leftarrow DR, SC \leftarrow 0$	Write, Sel=3 CLR_{SC}
	D_6T_6DRZ:	$PC \leftarrow PC+1$	INR_{PC}
Reg-ref	D_7IT_3	$= r, IR(i) = B_i$ (i = 0, 1, 2, ... , 11)	
	r:	$SC \leftarrow 0$	CLR_{SC}
CLA	rB_{11}:	$AC \leftarrow 0$	CLR_{AC}
CLE	rB_{10}:	$E \leftarrow 0$	CLR_E
CMA	rB_9:	$AC \leftarrow AC'$	CMA
CME	rB_8:	$E \leftarrow E'$	CME
CIR	rB_7:	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	LD_{AC}, SHR
CIL	rB_6:	$AC \leftarrow shl AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	LD_{AC}, SHL
INC	rB_5:	$AC \leftarrow AC + 1$	INR_{AC}
SPA	$rB_4AC(15)'$:	$PC \leftarrow PC + 1$	INR_{PC}
SNA	$rB_3AC(15)$:	$PC \leftarrow PC + 1$	INR_{PC}
SZA	rB_2ACZ:	$PC \leftarrow PC + 1$	INR_{PC}
SZE	rB_1EZ:	$PC \leftarrow PC + 1$	INR_{PC}
HLT	rB_0:	$S \leftarrow 0$	
In-out	D_7IT_3	$= p, IR(i) = B_i$ (i = 6, 7, 8, 9, 10, 11)	
	p:	$SC \leftarrow 0$	
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$	
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$	
SKI	pB_9FGI:	$PC \leftarrow PC + 1$	INR_{PC}
SKO	pB_8FGO:	$PC \leftarrow PC + 1$	INR_{PC}
ION	pB_7:	$IEN \leftarrow 1$	
IOF	pB_6:	$IEN \leftarrow 0$	



LD_{PC}
INR_{PC} ~~CLR_{PC}~~

LD_{AC}
INR_{AC}
CLR_{AC}

LD_{OUTR}

Read =

CLR_{SC}

LD_I

S_2-S_0

1a. AR Control Logic

RTL ASM#2

Fetch	$R'T_0$:	$AR \leftarrow PC$	1.
	$R'T_1$:	$IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$	
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{-}14)$, $AR \leftarrow IR(0\text{-}11)$, $I \leftarrow IR(15)$	2.
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$	3.
Interrupt:			
	$T_0T_1T_2$	$IEN(FGI + FGO)$: $R \leftarrow 1$	
	RT_0:	$AR \leftarrow 0$, $TR \leftarrow PC$	4.
	RT_1:	$M[AR] \leftarrow TR$, $PC \leftarrow 0$	
	RT_2:	$PC \leftarrow PC+1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$	
Memory-reference:			
AND	D_0T_4:	$DR \leftarrow M[AR]$	
	D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$	
		.....	
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$	5.

Scan Table 5-6 & find all the RTL statements in which AR is modified (is **destination** of RTLs).

- The first three such RTL statements are performed by enabling the LD bit of AR, i.e., LD(AR).
- The 4-th is done by activating the CLR
- The 5-fth statement is performed by enabling the INR bit of AR, i.e., INR(AR).

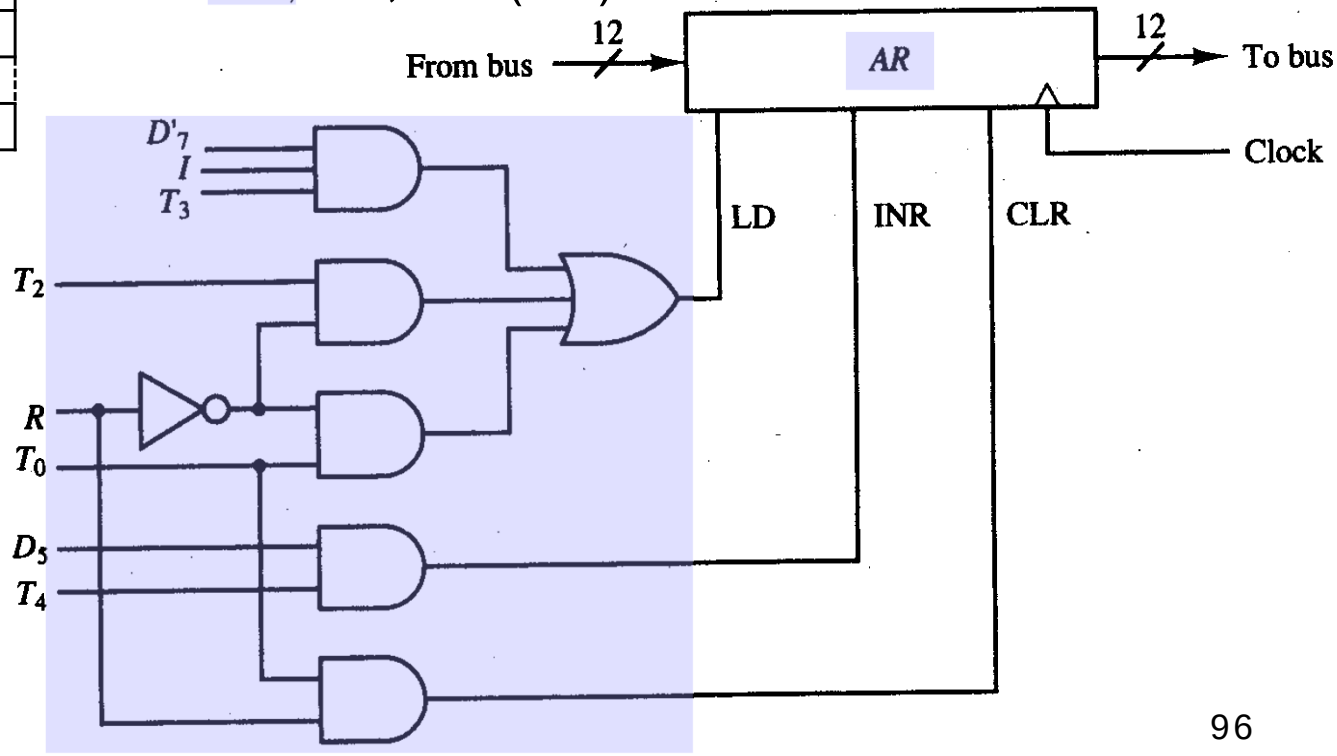
1. $R'T_0 : AR \leftarrow PC$
2. $R'T_2 : AR \leftarrow IR(0\text{-}11)$
3. $D'_7IT_3 : AR \leftarrow M[AR]$
4. $RT_0 : AR \leftarrow 0$
5. $D_5T_4 : AR \leftarrow AR + 1$

ASM#5

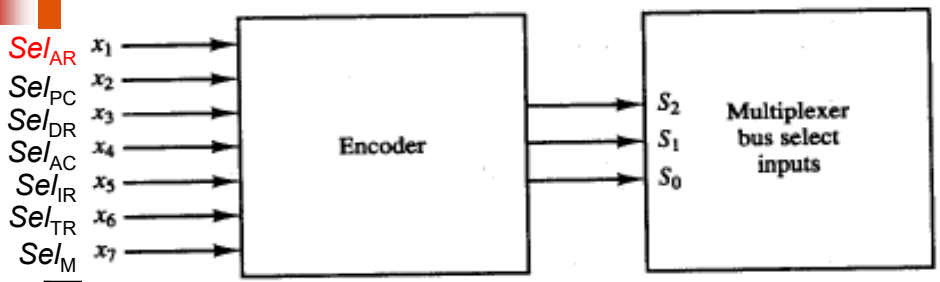
$LD(AR) = R'T_0 + R'T_2 + D'_7IT_3$

$CLR(AR) = RT_0$

$INR(AR) = D_5T_4$



Bus Control

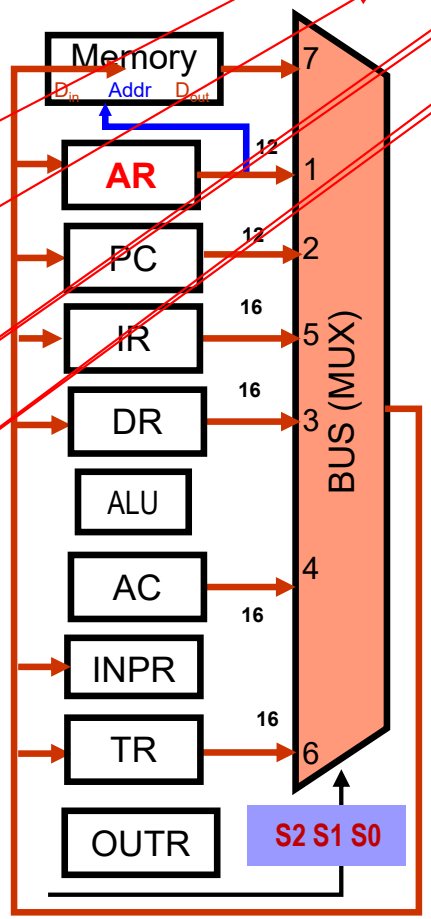


Reg selected for bus	x_1 Sel_{AR}	x_2 Sel_{PC}	x_3 Sel_{DR}	x_4 Sel_{AC}	x_5 Sel_{IR}	x_6 Sel_{TR}	x_7 Sel_{M}	S_2	S_1	S_0
none	**0**	0	0	0	0	0	0	0	0	0
AR	**1**	0	0	0	0	0	0	0	0	1
PC	0	1	0	0	0	0	0	0	1	0
DR	0	0	1	0	0	0	0	0	1	1
AC	0	0	0	1	0	0	0	1	0	0
IR	0	0	0	0	1	0	0	1	0	1
TR	0	0	0	0	0	1	0	1	1	0
M	0	0	0	0	0	0	1	1	1	1

$S_0 = x_1 + x_3 + x_5 + x_7$
$S_1 = x_2 + x_3 + x_6 + x_7$
$S_2 = x_4 + x_5 + x_6 + x_7$

To find x_1, **scan** the table of microoperations of Basic Computer from Table 5-6 and collect all the RTL statements where **AR** is a **Source Register** to be selected by the Bus MUX (on the *right side* of RTL!)

AR:	D_4T_4: $PC \leftarrow AR$ D_5T_5: $PC \leftarrow AR$	$Sel_{AR} = x_1 = D_4T_4 + D_5T_5$
PC:		$Sel_{PC} = x_2 =$
DR:		$Sel_{DR} = x_3 =$
AC:		$Sel_{AC} = x_4 =$
IR :		$Sel_{IR} = x_5 =$
TR:		$Sel_{TR} = x_6 =$
M :		$Sel_{M} = x_7 = R'T_1 + D'_7IT_3 + (D_0 + D_1 + D_2 + D_6)T_4$



Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{-}14)$,
		$AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$
Interrupt	$IEN\ (FGI+FGO)$:	$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC+1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$
Execute Memory-reference Instructions (MRI)		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$
STA.	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR,$
	D_6T_6DR':	$if\ (DR = 0)\ then\ (PC \leftarrow PC + 1), SC \leftarrow 0$
Execute Register-reference Instructions (RRI)		
	$D_7I'T_3 = r$	(common to all Register-ref. instructions)
		$IR(i) = B_i$ (i = 0, 1, 2, ... , 11)
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shl\ AC, AC(0) \leftarrow E, E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	$rB_4AC'(15)$:	$If\ (AC(15) = 0)\ then\ (PC \leftarrow PC + 1)$
SNA	$rB_3AC(15)$:	$If\ (AC(15) = 1)\ then\ (PC \leftarrow PC + 1)$
SZA	rB_2AC':	$If\ (AC = 0)\ then\ (PC \leftarrow PC + 1)$
SZE	rB_1E':	$If\ (E = 0)\ then\ (PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
Execute Input-output (IOI)		
	$D_7IT_3 = p$	(common to all input-output instructions)
	$IR(i) = B_i$	(i = 6, 7, 8, 9, 10, 11)
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$
SKI	pB_9FGI:	$If\ (FGI = 1)\ then\ (PC \leftarrow PC + 1)$
SKO	pB_8FGO:	$If\ (FGO = 1)\ then\ (PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

1b & 2b Registers Control Logic

Extend the RTL table with signals columns:

			$S_2S_1S_0$ source	Signals for selecting register **destination**
Fetch	$R'T_0$:	$AR \leftarrow PC$	010	**LD_AR**
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$	111	Read, LD_IR, INC_PC
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{-}14)$, $AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$	101	**LD_AR**, LD_I
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$	111	Read, **LD_AR**
Interrupt:				
$T_0T_1T_2IEN(FGI + FGO)$:		$R \leftarrow 1$		S_R,
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$	010	CL_AR, LD_TR,
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$	110	Write, CL_PC
	RT_2:	$PC \leftarrow PC+1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$		INC_PC, R_{SC}, R_R, R_{IEN},
Memory-reference:				
AND	D_0T_4:	$DR \leftarrow M[AR]$		
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$		
ADD	D_1T_4:	$DR \leftarrow M[AR]$		
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$		
LDA	D_2T_4:	$DR \leftarrow M[AR]$		
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$		
STA.	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$		
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$		
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$		
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$		
ISZ	D_6T_4:	$DR \leftarrow M[AR]$		
	D_6T_5:	$DR \leftarrow DR + 1$		
	D_6T_6:	$M[AR] \leftarrow DR$,		
	D_6T_6DR':	if $(DR = 0)$ then $(PC \leftarrow PC + 1), SC \leftarrow 0$		

RTL ASM#2

Control Signals ASM#4

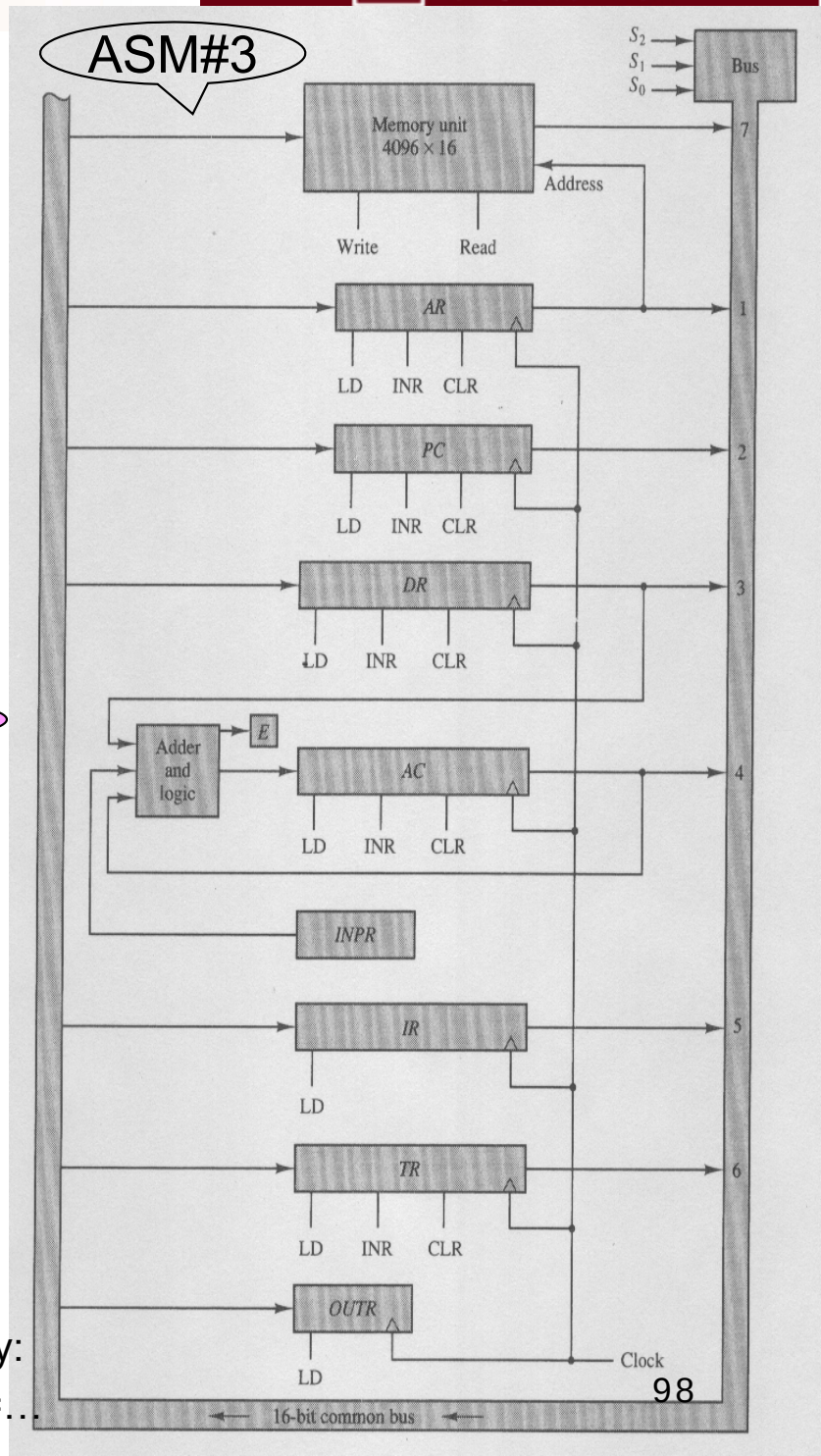
ASM#5

$LD\_AR = R'T_0 + R'T_2 + D'_7IT_3$; $CLR\_AR = RT_0$; $INR\_AR = D_5T_4$

S_2, S_1 and S_0 implemented without encoder, but with gates only:

$S_2 = R'T_1 + R'T_2 + D'_7IT_3 + RT_1$; $S_1 = R'T_0 + R'T_1 + D'_7IT_3 + RT_0 + RT_1$; $S_0 = \ldots$

ASM#3



1c & 2b Registers Control Logic

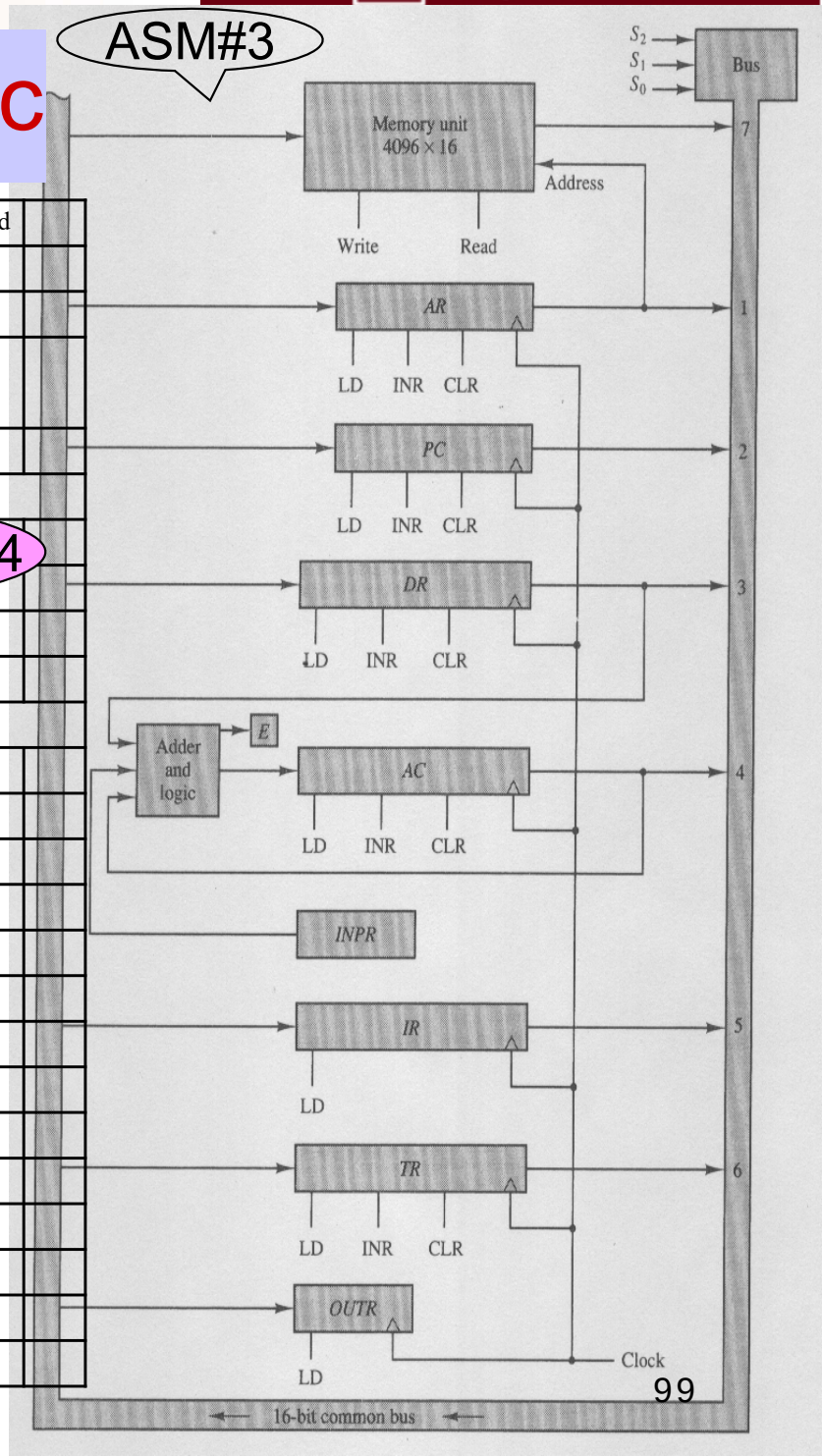
RTL ASM#2

			$S_2S_1S_0$	LD_{AR}	LD_{IR}	INC_{PC}	LD_{TR}	CLR_{PC}	LD_{DR}	read
Fetch	$R'T_0$:	$AR \leftarrow PC$	010	1						
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$	111		1	1				1
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{-}14)$,								
		$AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$	101	1	1					
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$	111	1						1
Interrupt:										
$T_0T_1T_2IEN(FGI + FGO)$:		$R \leftarrow 1$								
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$								
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$								
	RT_2:	$PC \leftarrow PC+1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$								
		Memory-reference:								
AND	D_0T_4:	$DR \leftarrow M[AR]$								
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$								
ADD	D_1T_4:	$DR \leftarrow M[AR]$								
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$								
LDA	D_2T_4:	$DR \leftarrow M[AR]$								
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$								
STA.	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$								
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$								
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$								
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$								
ISZ	D_6T_4:	$DR \leftarrow M[AR]$								
	D_6T_5:	$DR \leftarrow DR + 1$								
	D_6T_6:	$M[AR] \leftarrow DR$,								
	D_6T_6DR':	if $(DR = 0)$ then $(PC \leftarrow PC + 1), SC \leftarrow 0$								

Control Signals ASM#4

ASM#5

ASM#3



$LD\_AR = R'T_0 + R'T_2 + D'_7IT_3 \ldots$

uOttawa

Control Logic Comprehensive

Control Signals ASM#4

RTL ASM#2

ASM#5

			2	1																	3		5
			From $S_2S_1S_0$	To	LD AR	INR AR	CLR AR	LD PC	INR PC	CLR PC	LD IR	LD DR	INR DR	CLR DR	LD TR	INR TR	CLR TR	LD AC	INR AC	CLR AC	Write	Read	ALU
Fetch	$R'T_0$:	$AR \leftarrow PC$	010	LD_{AR}	1																		
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$	111	LD_{IR}					1		1											1	
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode IR(12-14), $AR \leftarrow IR(0\text{-}11), I \leftarrow IR(15)$	101	LD_{AR}	1																		
Indirect	D'_7IT_3	$AR \leftarrow M[AR]$	111	LD_{AR}	1																	1	
Interrupt	$T_0T_1T_2$	$IEN(FGI + FGO): R \leftarrow 1$		S_R																			
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$	010	LD_{TR}			1																
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$	110	Write						1													
	RT_2:	$PC \leftarrow PC+1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$	R_{IEN}	INR_{PC}					1														
Execute		**Memory-reference:**																					
AND	D_0T_4:	$DR \leftarrow M[AR]$	111	LD_{DR}								1										1	
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$		LD_{AC}														1					AND
ADD	D_1T_4:	$DR \leftarrow M[AR]$	111	LD_{DR}								1										1	
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$		LD_{AC}														1					ADD
LDA	D_2T_4:	$DR \leftarrow M[AR]$	111	LD_{DR}								1										1	
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$		LD_{AC}														1					DR
STA.	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$	100	Write																			
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$	001	LD_{PC}				1													1		
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$	010	Write		1																	
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$	001	LD_{PC}				1													1		
ISZ	D_6T_4:	$DR \leftarrow M[AR]$	111	LD_{DR}								1										1	
	D_6T_5:	$DR \leftarrow DR + 1$		INR_{DR}									1										
	D_6T_6:	$M[AR] \leftarrow DR,$	011	Write																			
	D_6T_6DR'	$PC \leftarrow PC + 1, SC \leftarrow 0$		INR_{PC}					1														

$LD\_AR = R'T_0 + R'T_2 + D'_7IT_3$; $CLR\_AR = RT_0$; $INR\_AR = D_5T_4$

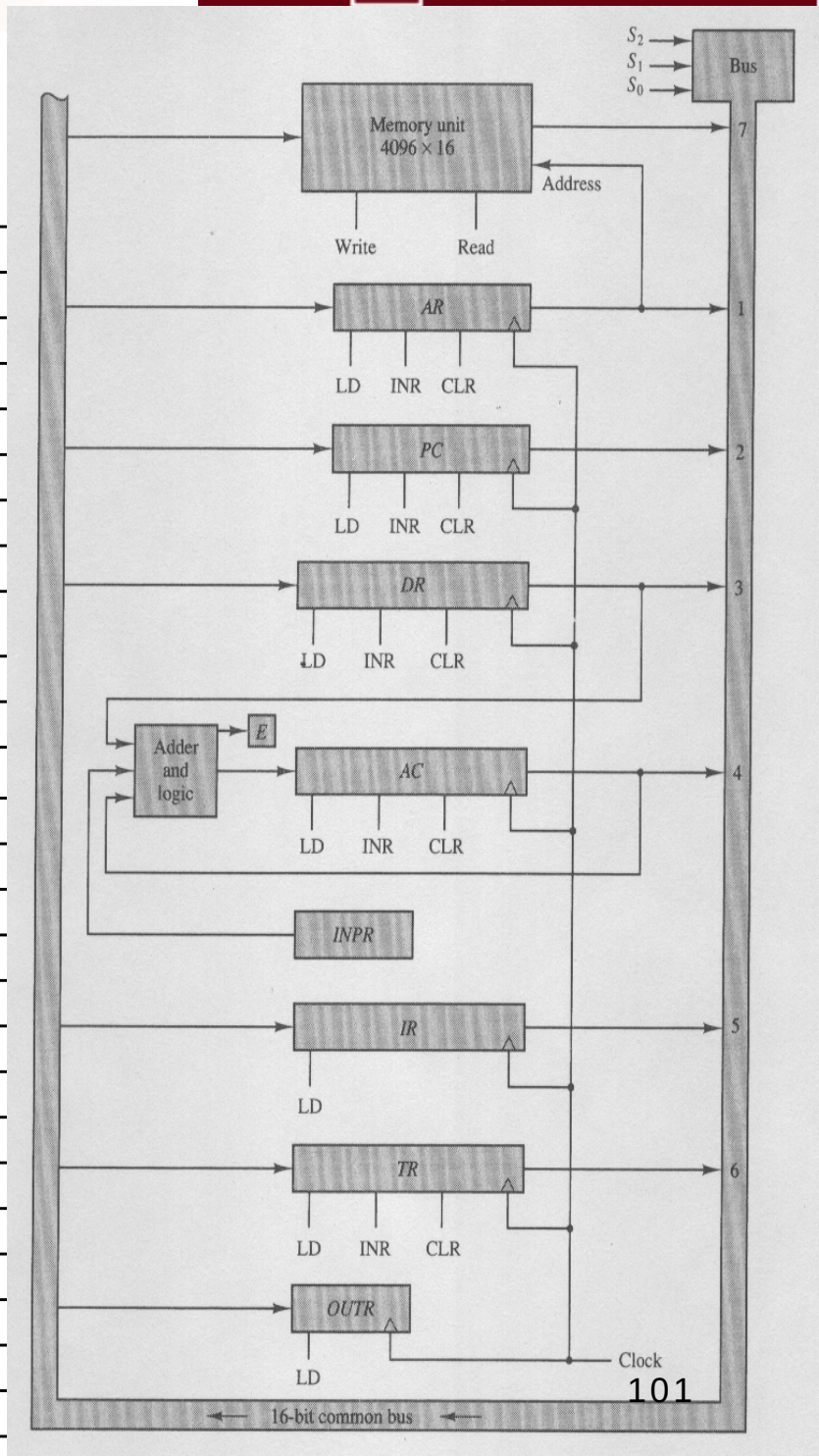
uOttawa

4. FF Control Logic

		Register-reference:	R_{IEN}	S_{IEN}				
	$D_7I'T_3 = r$	(common to all RRI)						
	$IR(i) = B_i$	(i = 0, 1, 2, ... , 11)						
	r:	SC ← 0						
CLA	rB_{11}:	$AC \leftarrow 0$						
CLE	rB_{10}:	$E \leftarrow 0$						
CMA	rB_9:	$AC \leftarrow AC'$						
CME	rB_8:	$E \leftarrow E'$						
CIR	rB_7:	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$						
CIL	rB_6:	$AC \leftarrow shlAC, AC(0) \leftarrow E, E \leftarrow AC(15)$						
INC	rB_5:	$AC \leftarrow AC + 1$						
SPA	$rB_4AC'(15)$	$If\ (AC(15) = 0)\ then\ (PC \leftarrow PC + 1)$						
SNA	rB_3AC(15):	$If\ (AC(15) = 1)\ then\ (PC \leftarrow PC + 1)$						
SZA	rB_2AC':	$If\ (AC = 0)\ then\ (PC \leftarrow PC + 1)$						
SZE	rB_1E':	$If\ (E = 0)\ then\ (PC \leftarrow PC + 1)$						
HLT	rB_0:	$S \leftarrow 0$						
		Input-output:						
	$D_7IT_3 = p$	(common to all IO instructions)						
	$IR(i) = B_i$	(i = 6, 7, 8, 9, 10, 11)						
	p:	$SC \leftarrow 0$						
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$						
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$						
SKI	pB_9FGI:	$If\ (FGI = 1)\ then\ (PC \leftarrow PC +$						
SKO	pB_8FGO:	$If\ (FGO = 1)\ then\ (PC \leftarrow PC + 1)$						
ION	pB_7:	***IEN ← 1***		1				
IOF	pB_6:	***IEN ← 0***	1					

Control Signals ASM#4

RTL ASM#2



RTL ASM#2

4. IEN FF Control Logic

- For example, Table 5-6 (of the textbook) shows that FF IEN is loaded as follows
  - pB_7: $IEN \leftarrow 1$ $S_{IEN} = 1$ where $p = D_7 I T_3$, $B_7 = IR(7)$, and $B_6 = IR(6)$
  - pB_6: $IEN \leftarrow 0$ $R_{IEN1} = 1$
  - RT_2: $IEN \leftarrow 0$ $R_{IEN2} = 1$

Control Signals ASM#4

J^n	K^n	Q^{n+1}	Function
0	0	Q^n	Hold
0	1	0	Reset
1	0	1	Set
1	1	Q^n	Toggle

HOW TO USE A JK-FF TO IMPLEMENT IEN FF:

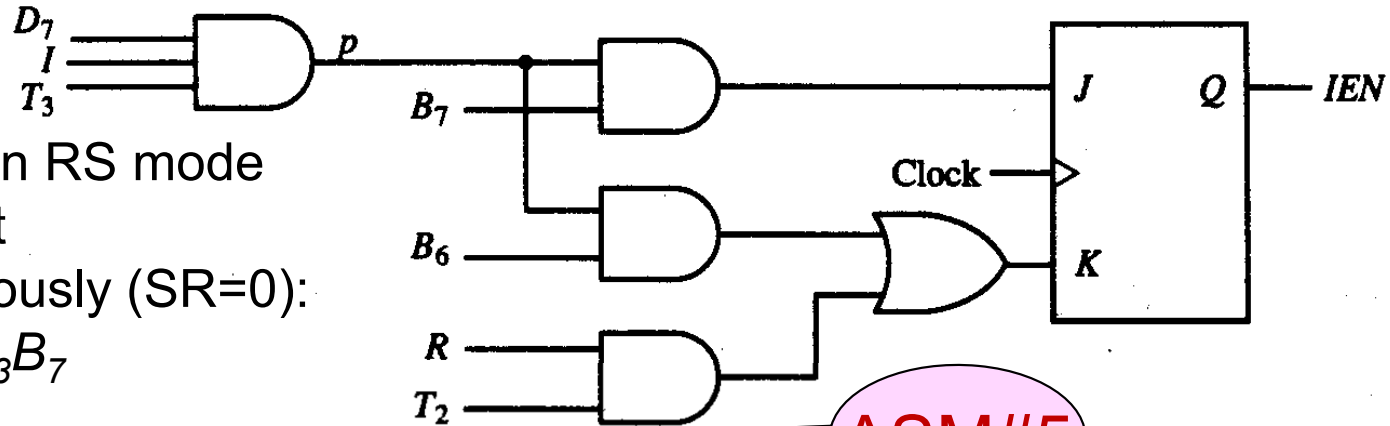
Do not include any term in J & K equations to keep the current state

To **reset** a JK-FF, add that term to the K equation, but not to J

To **set** a JK-FF, add that term to the J equation, but not to K

To **toggle** a JK-FF, add that term to both J & K equations

To load value "*a*" into a JK-FF, AND *a* with the term and include in the J equation, while "*a*' " is AND-ed with the term and it is applied to K. In particular to load whatever next state, set $a = Q^+$. The JK-FF input excitation equations (J & K) are derived in terms of the control signals.

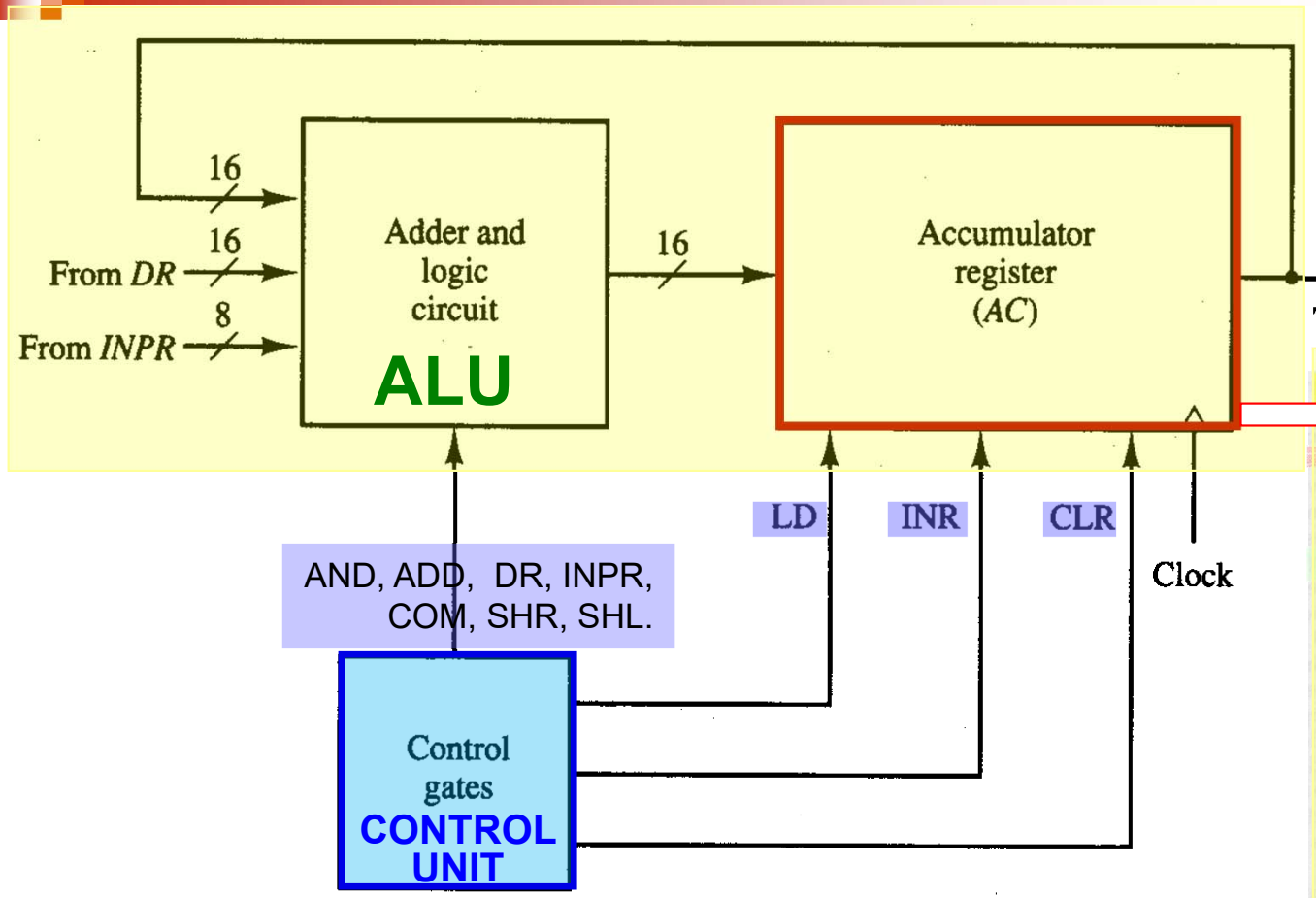


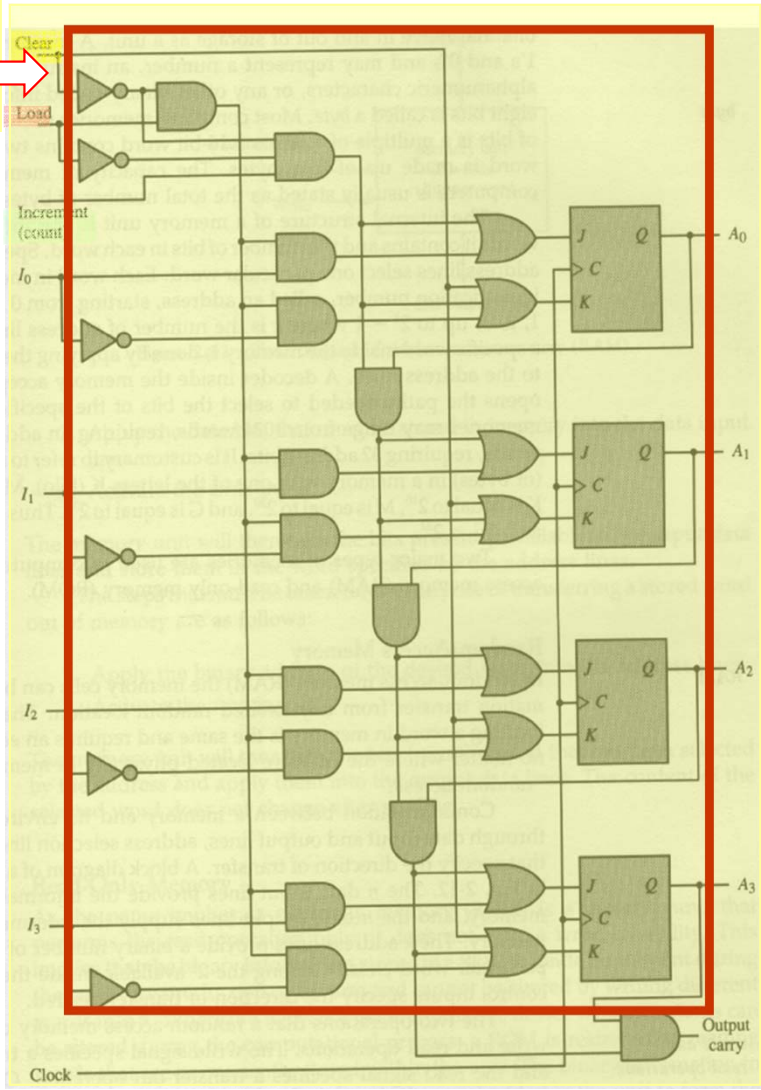
A JK flip-flop can be used in RS mode since here S and R are not required to be 1 simultaneously (SR=0):

$\boldsymbol{J_{IEN} = \Sigma(S_{IEN})} = pB_7 = D_7 I T_3 B_7$

$\boldsymbol{K_{IEN} = \Sigma(R_{IEN})} = pB_6 + RT_2$

ASM#5

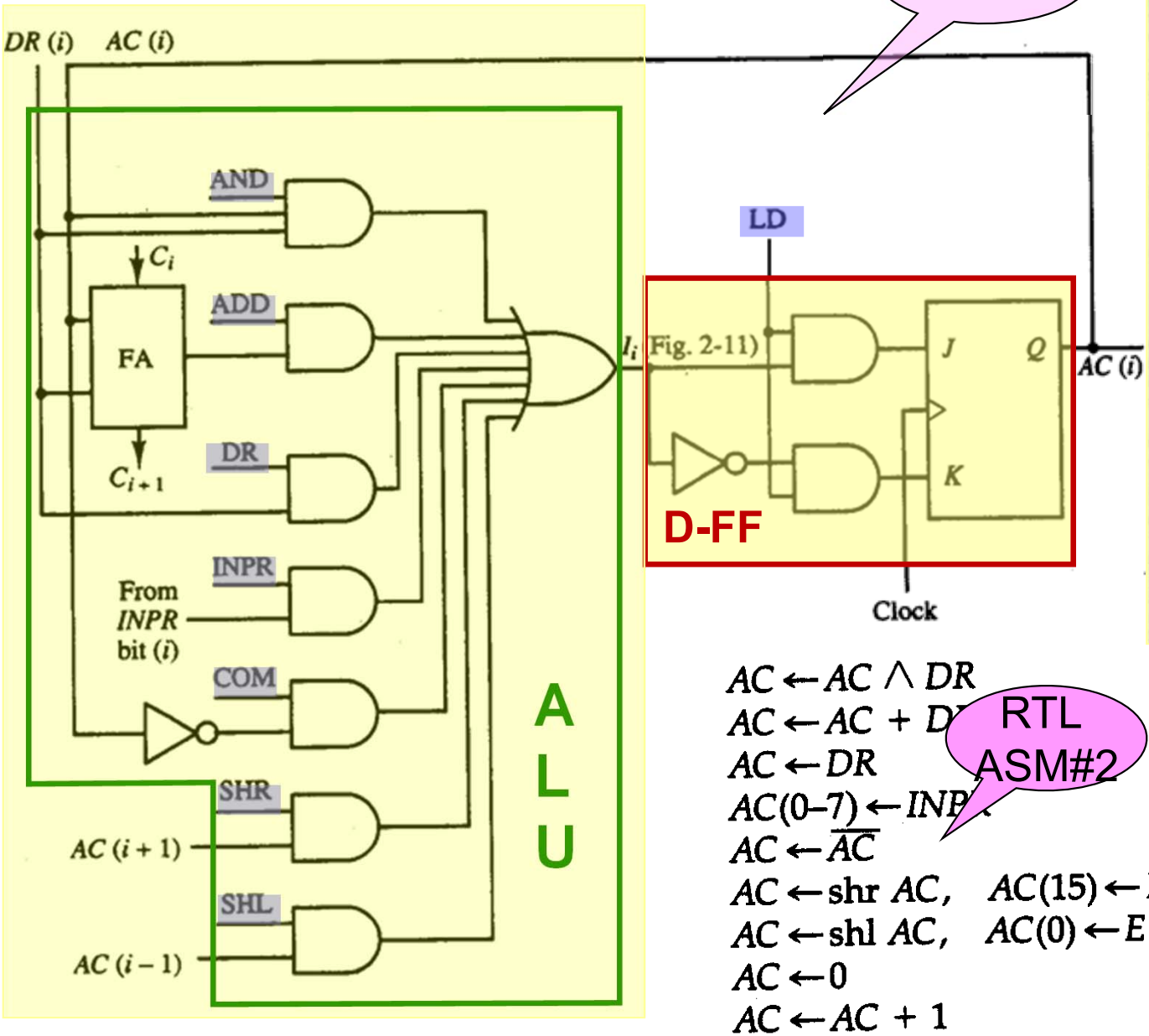


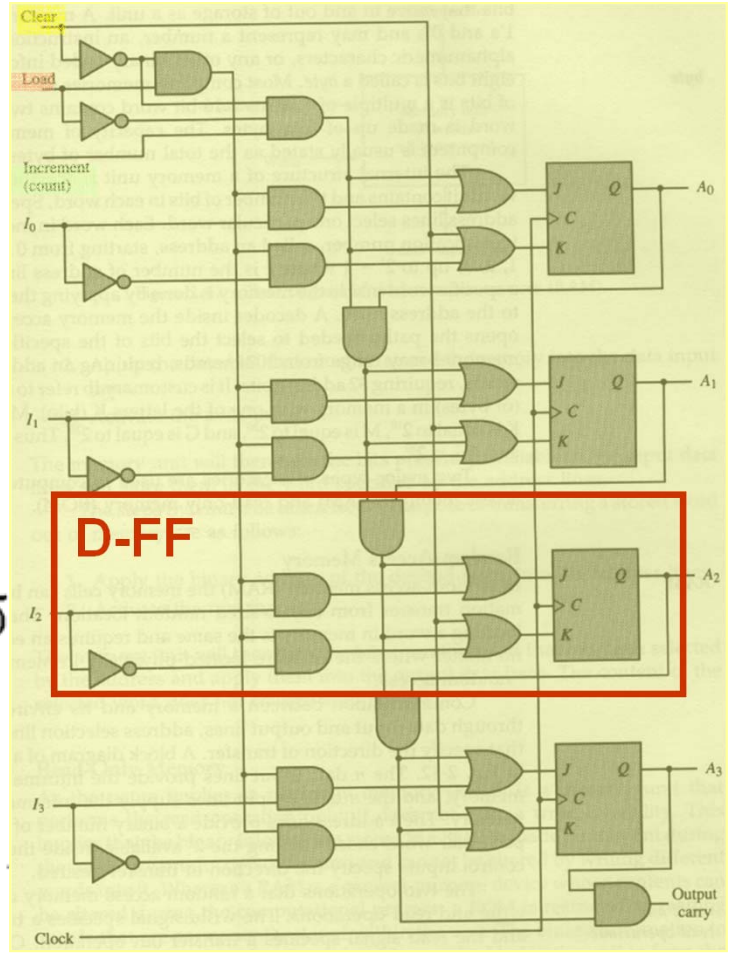


ASM#1

ALU Block Diagram

AC+ALU





RTL ASM#2

$AC \leftarrow AC \wedge DR$	AND with *DR*
$AC \leftarrow AC + DR$	Add with *DR*
$AC \leftarrow DR$	Transfer from *DR*
$AC(0\text{–}7) \leftarrow INPR$	Transfer from *INPR*
$AC \leftarrow \overline{AC}$	Complement
$AC \leftarrow \text{shr } AC, \quad AC(15) \leftarrow E$	Shift right
$AC \leftarrow \text{shl } AC, \quad AC(0) \leftarrow E$	Shift left
$AC \leftarrow 0$	Clear
$AC \leftarrow AC + 1$	Increment

5. ALU Control

To design the logic associated with AC, it is necessary to scan Table 5-6 (of the textbook) and extract all the statements in which AC is loaded:

RTL ASM#2

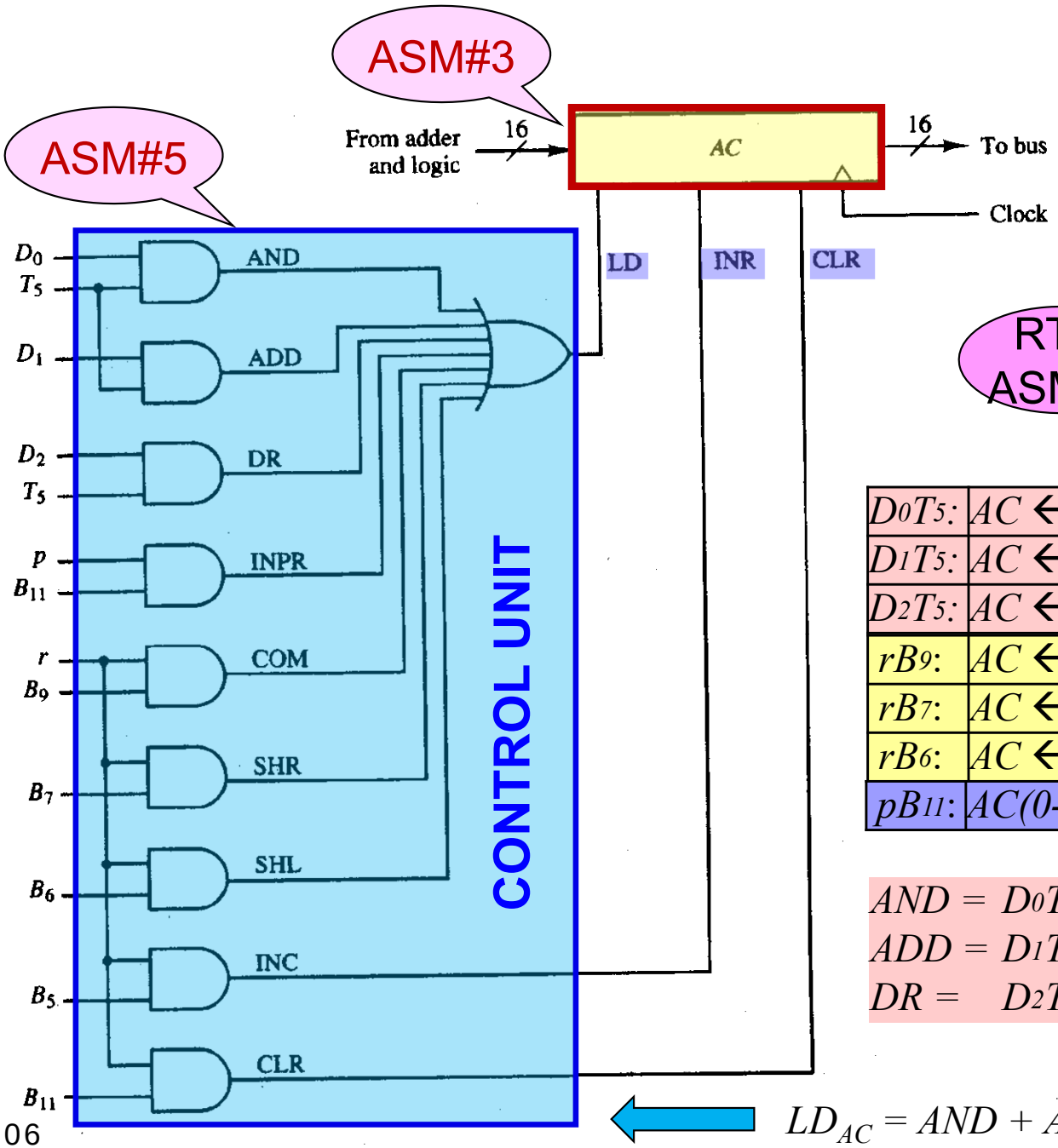
$AC \leftarrow AC \wedge DR$	AND with DR
$AC \leftarrow AC + DR$	Add with DR
$AC \leftarrow DR$	Transfer from DR
$AC(0\text{–}7) \leftarrow INPR$	Transfer from $INPR$
$AC \leftarrow \overline{AC}$	Complement
$AC \leftarrow shr\ AC$, $AC(15) \leftarrow E$	Shift right
$AC \leftarrow shl\ AC$, $AC(0) \leftarrow E$	Shift left
$AC \leftarrow 0$	Clear
$AC \leftarrow AC + 1$	Increment

ASM#4

D_0T_5:	$AC \leftarrow AC \wedge DR,\ SC \leftarrow 0$	AND
D_1T_5:	$AC \leftarrow AC + DR,\ E \leftarrow C_{out},\ SC \leftarrow 0$	ADD
D_2T_5:	$AC \leftarrow DR,\ SC \leftarrow 0$	DR
rB_9:	$AC \leftarrow AC'$	COM
rB_7:	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$	SHR
rB_6:	$AC \leftarrow shl\ AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$	SHL
pB_{11}:	$AC(0\text{-}7) \leftarrow INPR,\ FGI \leftarrow 0$	$INPR$

Fetch	$R'T_0$:	$AR \leftarrow PC$	*ALU*	*AC*
	$R'T_1$:	$IR \leftarrow M[AR],\ PC \leftarrow PC + 1$		
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow Decode\ IR(12\text{-}14),$		
		$AR \leftarrow IR(0\text{-}11),\ I \leftarrow IR(15)$		
Indirect	D'_7IT_3:	$AR \leftarrow M[AR]$		
Interrupt	$IEN\ (FGI+FGO)$: $R \leftarrow 1$			
	RT_0:	$AR \leftarrow 0,\ TR \leftarrow PC$		
	RT_1:	$M[AR] \leftarrow TR,\ PC \leftarrow 0$		
	RT_2:	$PC \leftarrow PC+1,\ IEN \leftarrow 0,\ R \leftarrow 0,\ SC \leftarrow 0$		
Execute Memory-reference Instructions (MRI)				
AND	D_0T_4:	$DR \leftarrow M[AR]$		
	D_0T_5:	$AC \leftarrow AC \wedge DR,\ SC \leftarrow 0$	AND	LD$_{AC}$
ADD	D_1T_4:	$DR \leftarrow M[AR]$		
	D_1T_5:	$AC \leftarrow AC + DR,\ E \leftarrow C_{out},\ SC \leftarrow 0$	ADD	LD$_{AC}$
LDA	D_2T_4:	$DR \leftarrow M[AR]$		
	D_2T_5:	$AC \leftarrow DR,\ SC \leftarrow 0$	DR	LD$_{AC}$
STA.	D_3T_4:	$M[AR] \leftarrow AC,\ SC \leftarrow 0$		
BUN	D_4T_4:	$PC \leftarrow AR,\ SC \leftarrow 0$		
BSA	D_5T_4:	$M[AR] \leftarrow PC,\ AR \leftarrow AR + 1$		
	D_5T_5:	$PC \leftarrow AR,\ SC \leftarrow 0$		
ISZ	D_6T_4:	$DR \leftarrow M[AR]$		
	D_6T_5:	$DR \leftarrow DR + 1$		
	D_6T_6:	$M[AR] \leftarrow DR,$		
	D_6T_6DR':	$if\ (DR = 0)\ then\ (PC \leftarrow PC + 1),\ SC \leftarrow 0$		
Execute Register-reference Instructions (RRI)				
	$D_7I'T_3 = r$	(common to all Register-ref. instructions)		
		$IR(i) = B_i$ (i = 0, 1, 2, ... , 11)		
	r:	$SC \leftarrow 0$		
CLA	rB_{11}:	$AC \leftarrow 0$		CLR$_{AC}$
CLE	rB_{10}:	$E \leftarrow 0$		
CMA	rB_9:	$AC \leftarrow AC'$	COM	LD$_{AC}$
CME	rB_8:	$E \leftarrow E'$		
CIR	rB_7:	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$	SHR	LD$_{AC}$
CIL	rB_6:	$AC \leftarrow shl\ AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$	SHL	LD$_{AC}$
INC	rB_5:	$AC \leftarrow AC + 1$		INC$_{AC}$
SPA	$rB_4AC'(15)$:	$If\ (AC(15) = 0)\ then\ (PC \leftarrow PC + 1)$		
SNA	$rB_3AC(15)$:	$If\ (AC(15) = 1)\ then\ (PC \leftarrow PC + 1)$		
SZA	rB_2AC':	$If\ (AC = 0)\ then\ (PC \leftarrow PC + 1)$		
SZE	rB_1E':	$If\ (E = 0)\ then\ (PC \leftarrow PC + 1)$		
HLT	rB_0:	$S \leftarrow 0$		
Execute Input-output (IOI)				
	$D_7IT_3 = p$	(common to all input-output instructions)		
	$IR(i) = B_i$	(i = 6, 7, 8, 9, 10, 11)		
	p:	$SC \leftarrow 0$		
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR,\ FGI \leftarrow 0$	$INPR$	LD$_{AC}$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7),\ FGO \leftarrow 0$		
SKI	pB_9FGI:	$If\ (FGI = 1)\ then\ (PC \leftarrow PC + 1)$		
SKO	pB_8FGO:	$If\ (FGO = 1)\ then\ (PC \leftarrow PC + 1)$		
ION	pB_7:	$IEN \leftarrow 1$		
IOF	pB_6:	$IEN \leftarrow 0$		

5. AC + ALU Control Logic (ASM#5)



RTL ASM#2 | Signals ASM#4

D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$	AND
D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$	ADD
D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$	DR
rB_9:	$AC \leftarrow AC'$	COM
rB_7:	$AC \leftarrow shr\ AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	SHR
rB_6:	$AC \leftarrow shl\ AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	SHL
pB_{11}:	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$	INPR

$AND = D_0T_5$
$ADD = D_1T_5$
$DR = D_2T_5$

$COM = rB_9$
$SHR = rB_7$
$SHL = rB_6$

$INPR = pB_{11}$

Signals ASM#5

$$LD_{AC} = AND + ADD + DR + COM + SHR + SHL + INPR$$

ASM
Design Example

Dr. Voicu Groza

SITE Hall, Room 5017
562 5800 ext. 2159
Groza@EECS.uOttawa.ca

uOttawa

L'Université canadienne
Canada's university

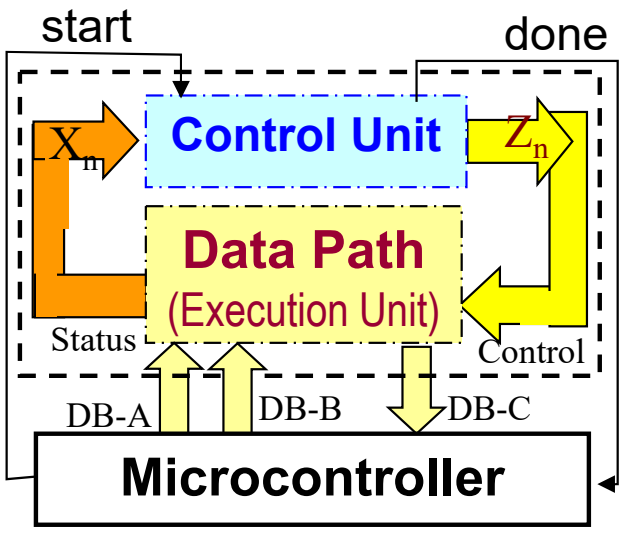
Université d'Ottawa | University of Ottawa

www.uOttawa.ca

ASM Design Example

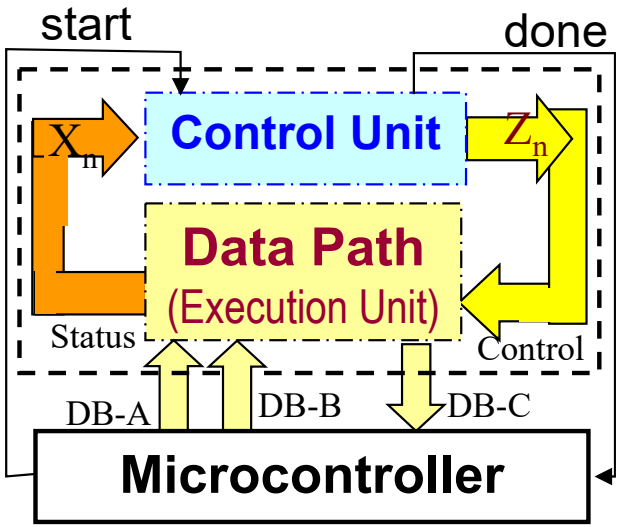
- The datapath of a dedicated coprocessor has three 16-bit registers *AR, BR,* and *CR*.
- When the coprocessor is triggered by a signal *start*, it will perform the following operations:
  - Transfer two 16-bit signed numbers (provided by the microcontroller over two data buses *DB-A* and *DB-B* in 2's-complement representation) to *AR* and *BR.*
  - If the number in *AR* is negative, divide the number in *AR* by 2 and transfer the result to register *CR.*
  - If the number in *AR* is positive but nonzero, multiply the number in *BR* by 2 and transfer the result to register *CR.*
  - If the number in *AR* is zero, clear register *CR* to 0.
  - Signalize the end of the processing by setting a flag *done*.

#1 Block Diagram

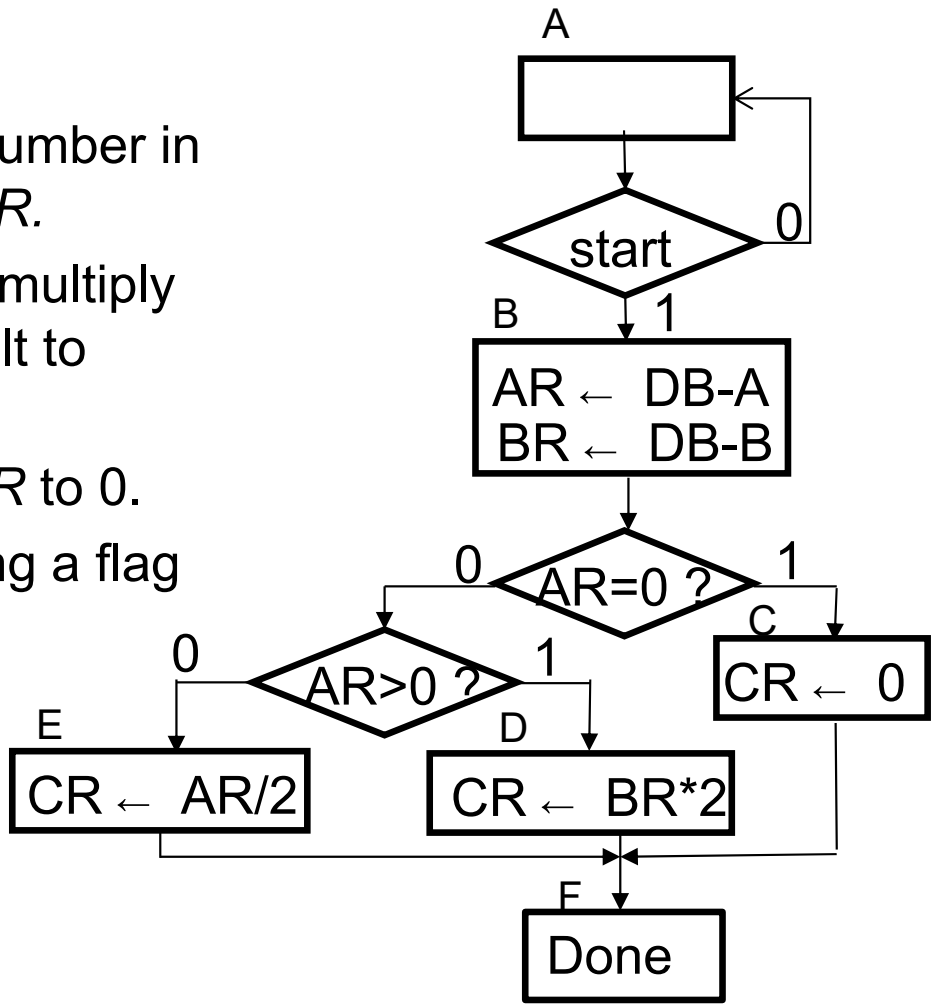


#1 Block Diagram

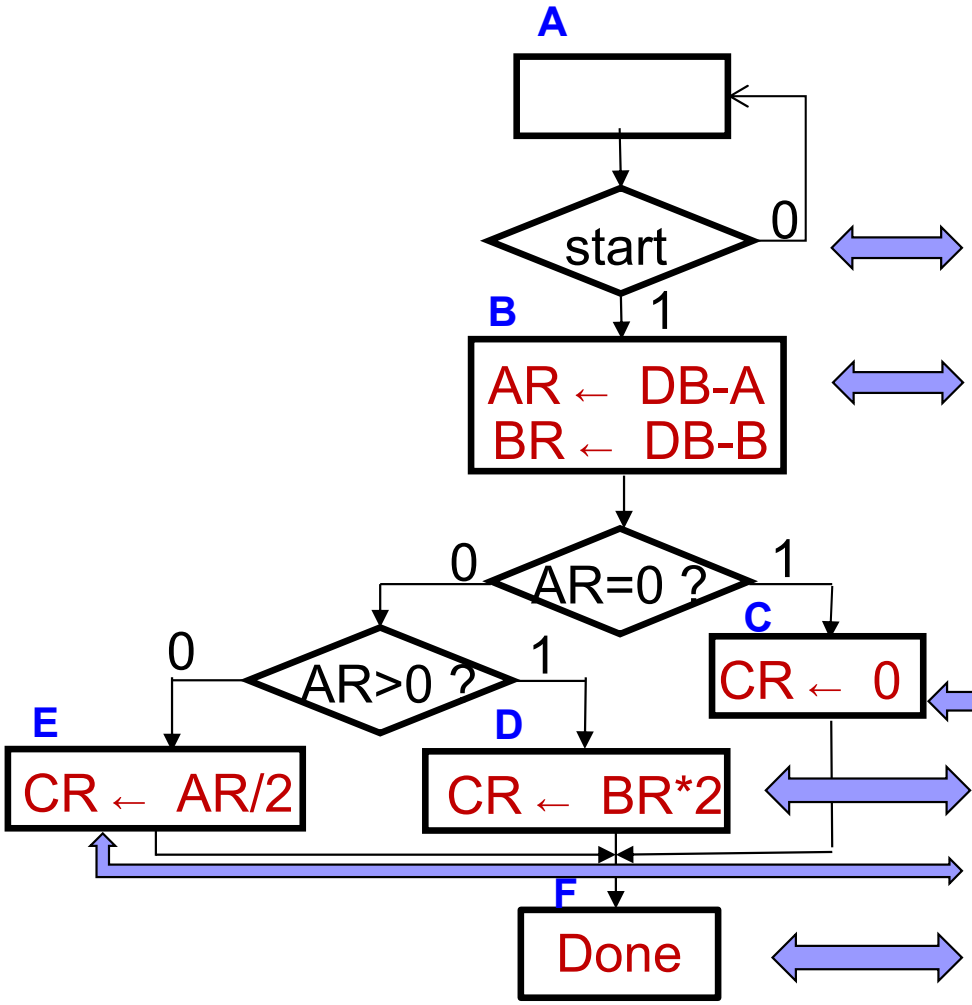
- Transfer two 16-bit signed numbers (provided by the microcontroller over two data buses *DB-A* and *DB-B* in 2's-complement representation) to *AR* & *BR.*
- If the number in *AR* is negative, divide the number in *AR* by 2 and transfer the result to register *CR.*
- If the number in *AR* is positive but nonzero, multiply the number in *BR* by 2 and transfer the result to register *CR.*
- If the number in *AR* is zero, clear register *CR* to 0.
- Signalize the end of the processing by setting a flag *done*.



ASM #2 **Flowchart** (High Level ***RTL***)



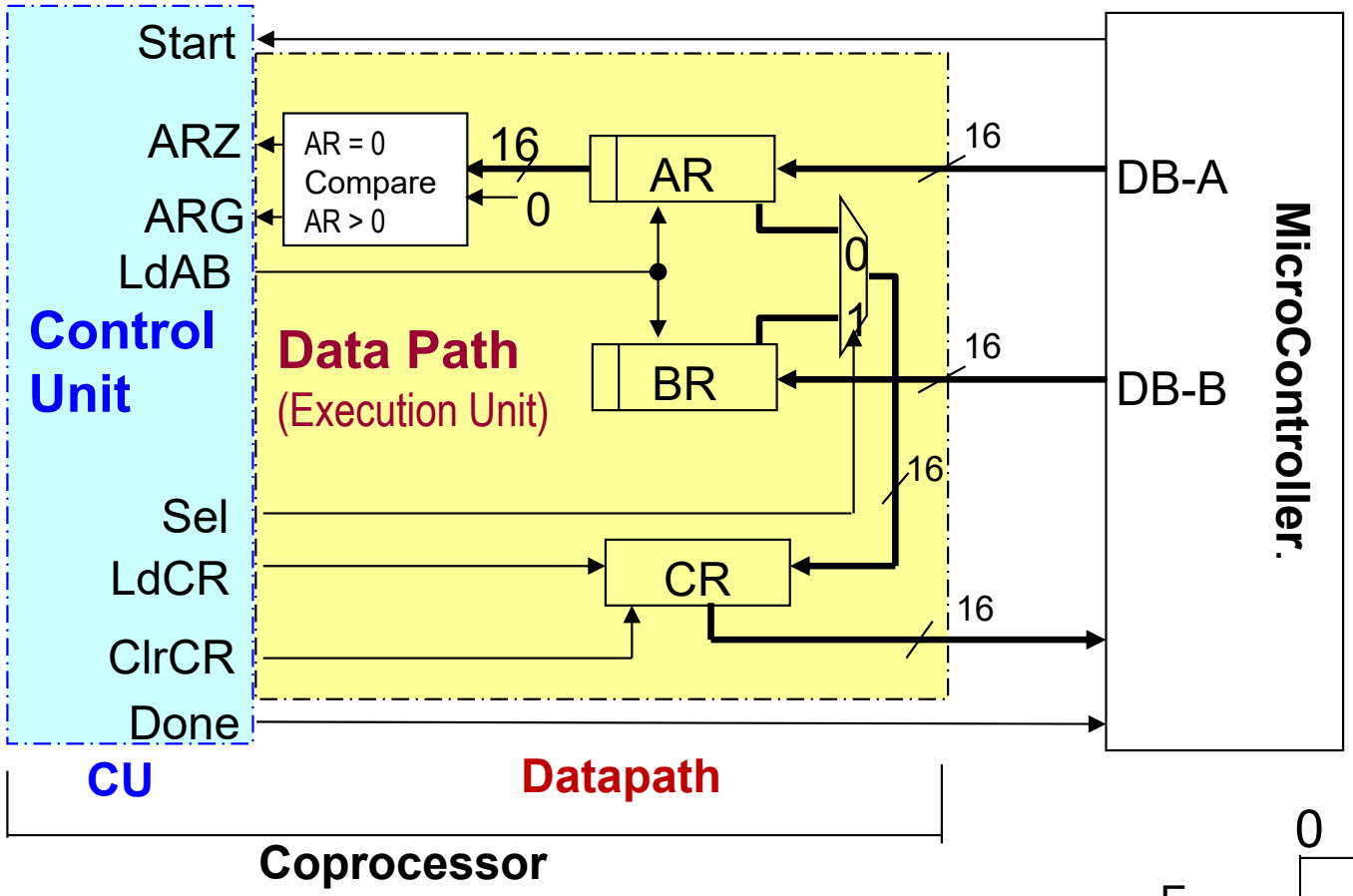
ASM #2 **Flowchart** (High Level ***RTL***)



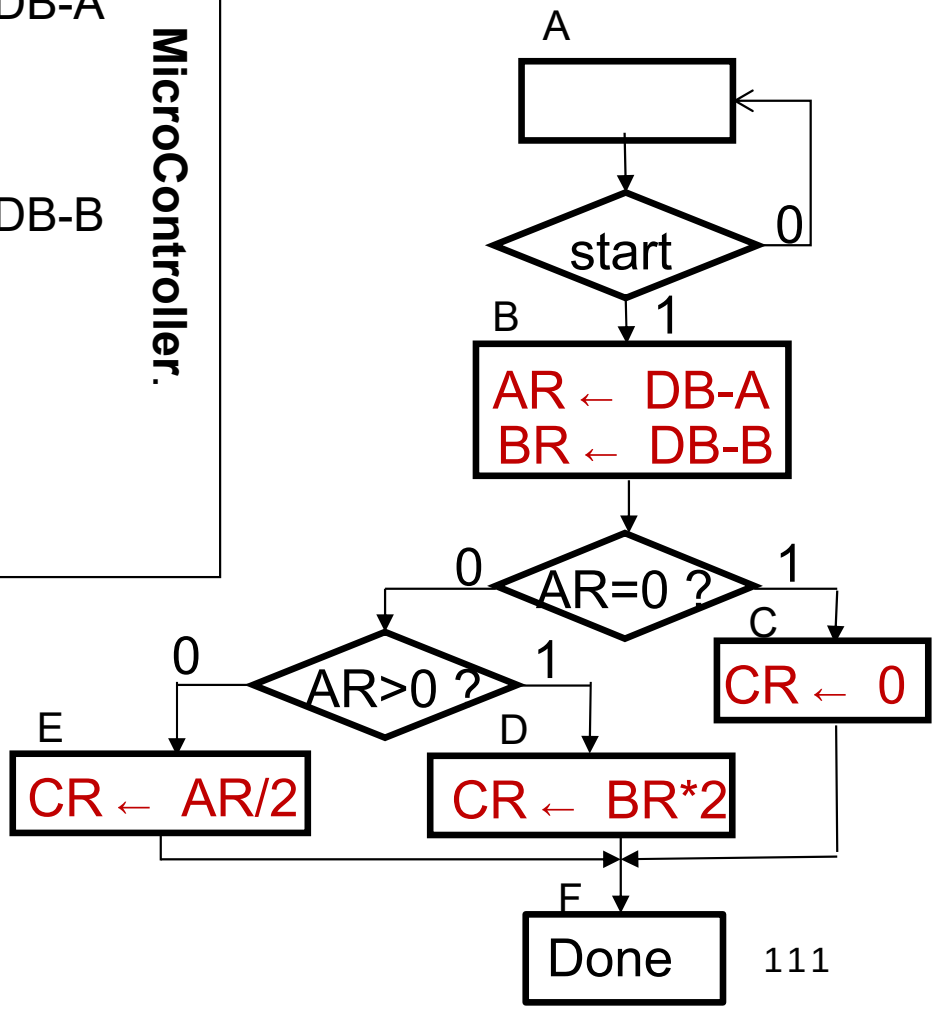
RTL

	$Z_n = \lambda(S_n)$	$S_{n+1} = \delta(S_n, X_n)$
A Start:		SC←B
B:	AR← DB-A BR←DB-B	
B (AR=0):		SC←C
B (AR≠0)(AR>0):		SC←D
B (AR≠0)(AR≤0):		SC←E
C:	CR←0	SC←F
D:	CR←BR*2	SC←F
E:	CR←AR/2	SC←F
F:		SC←A

ASM #3 Datapath

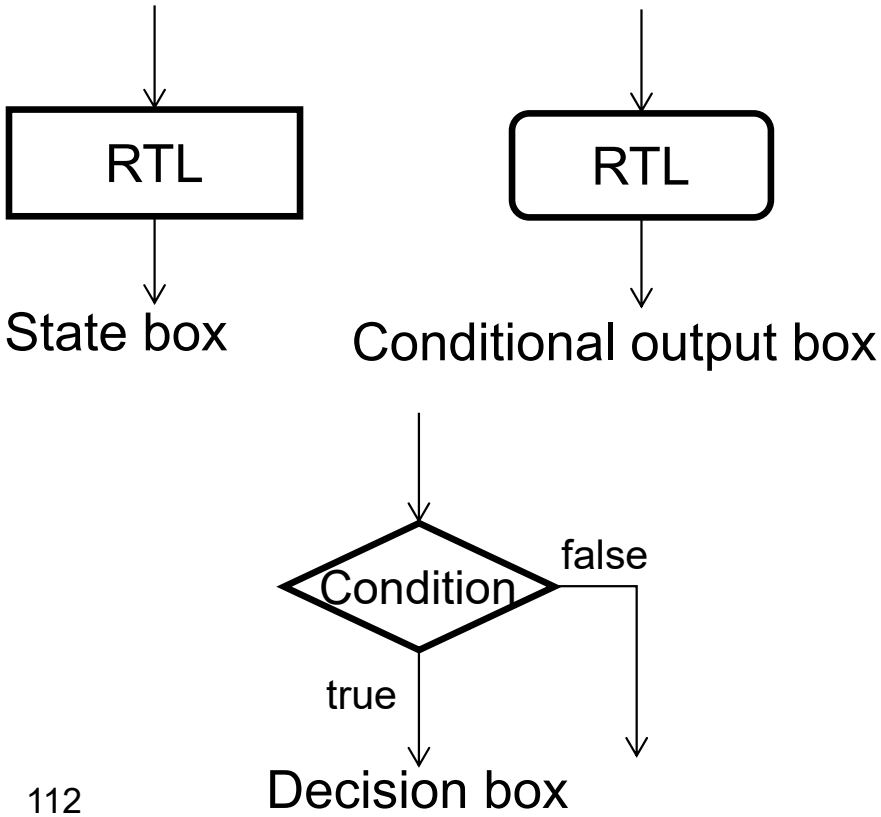


Determine the functional units (registers, buses, ALU's) needed to execute the microoperations described in the high-level ASM flowchart.

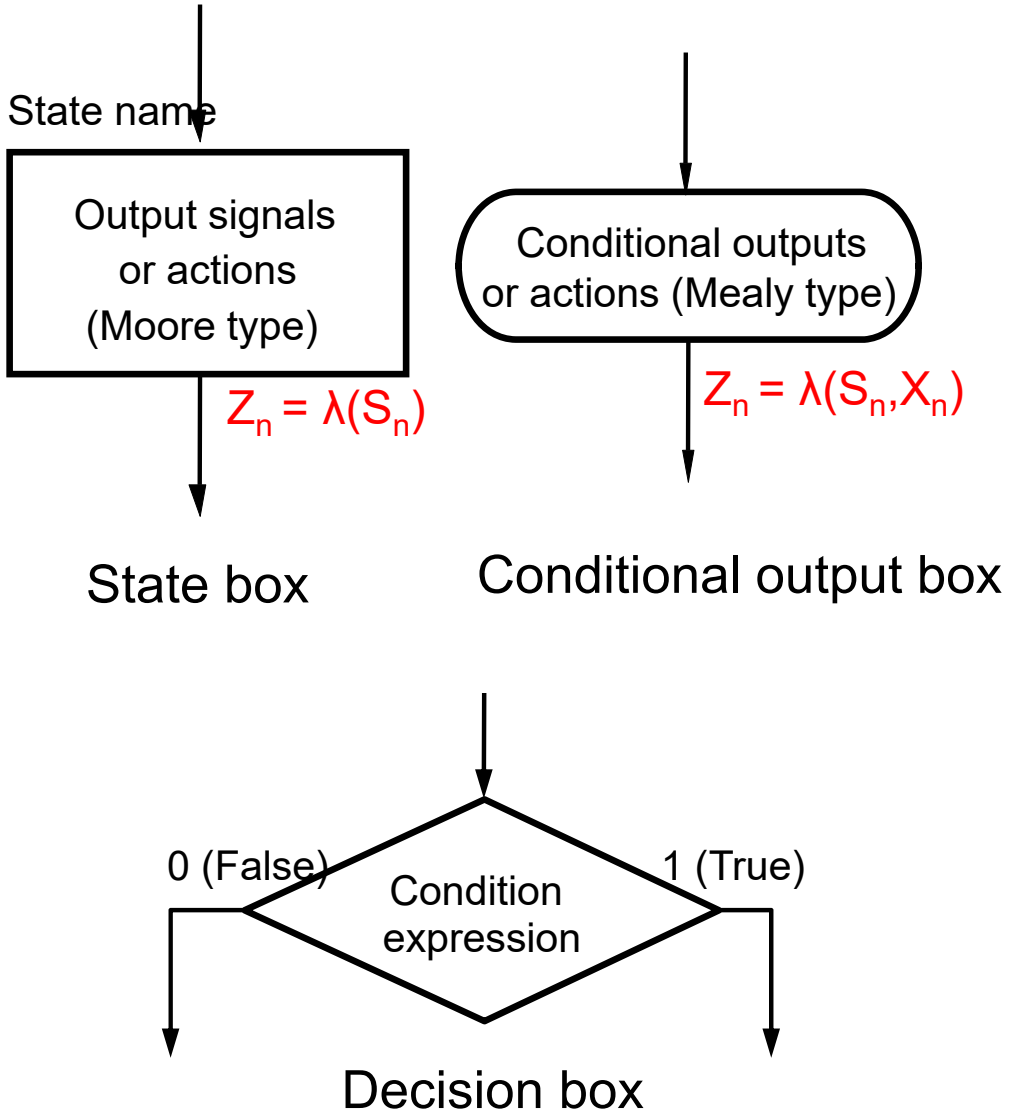


ASM#2 **Flowcharts**

ASM textual or tabular charts replace flowcharts because of their ease of modeling complexity and representing large synchronous sequential circuits.

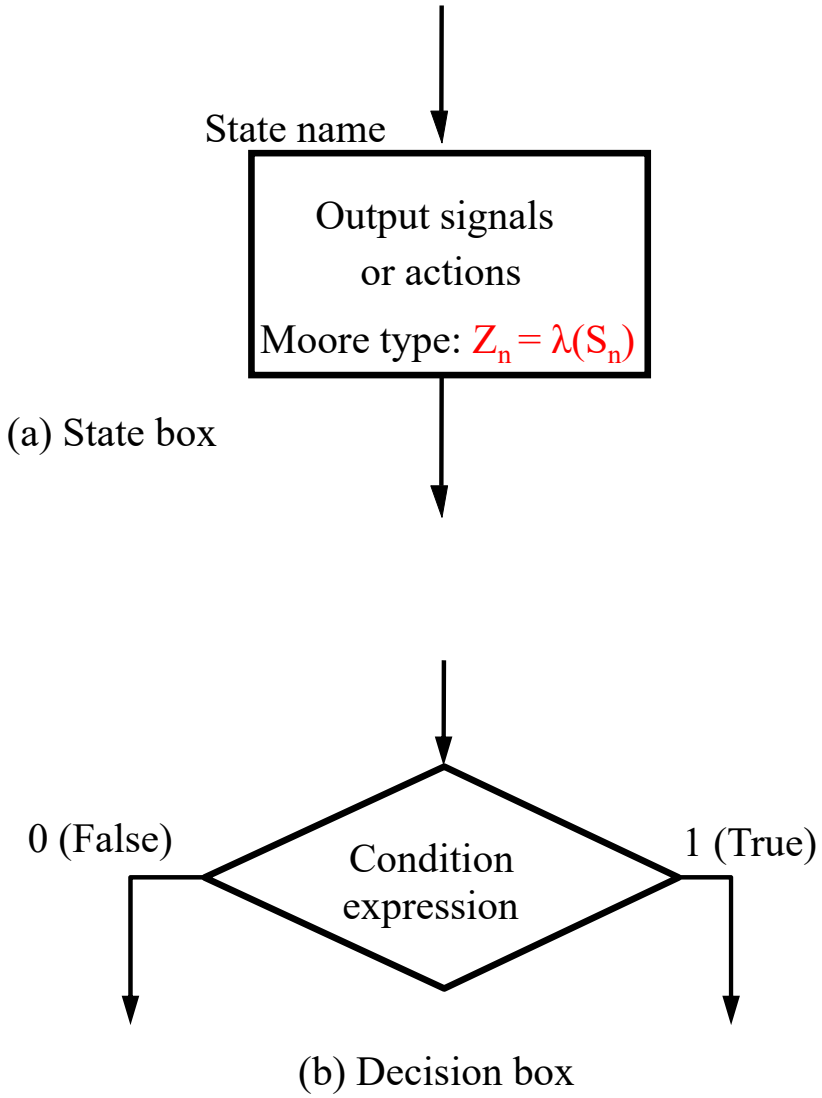


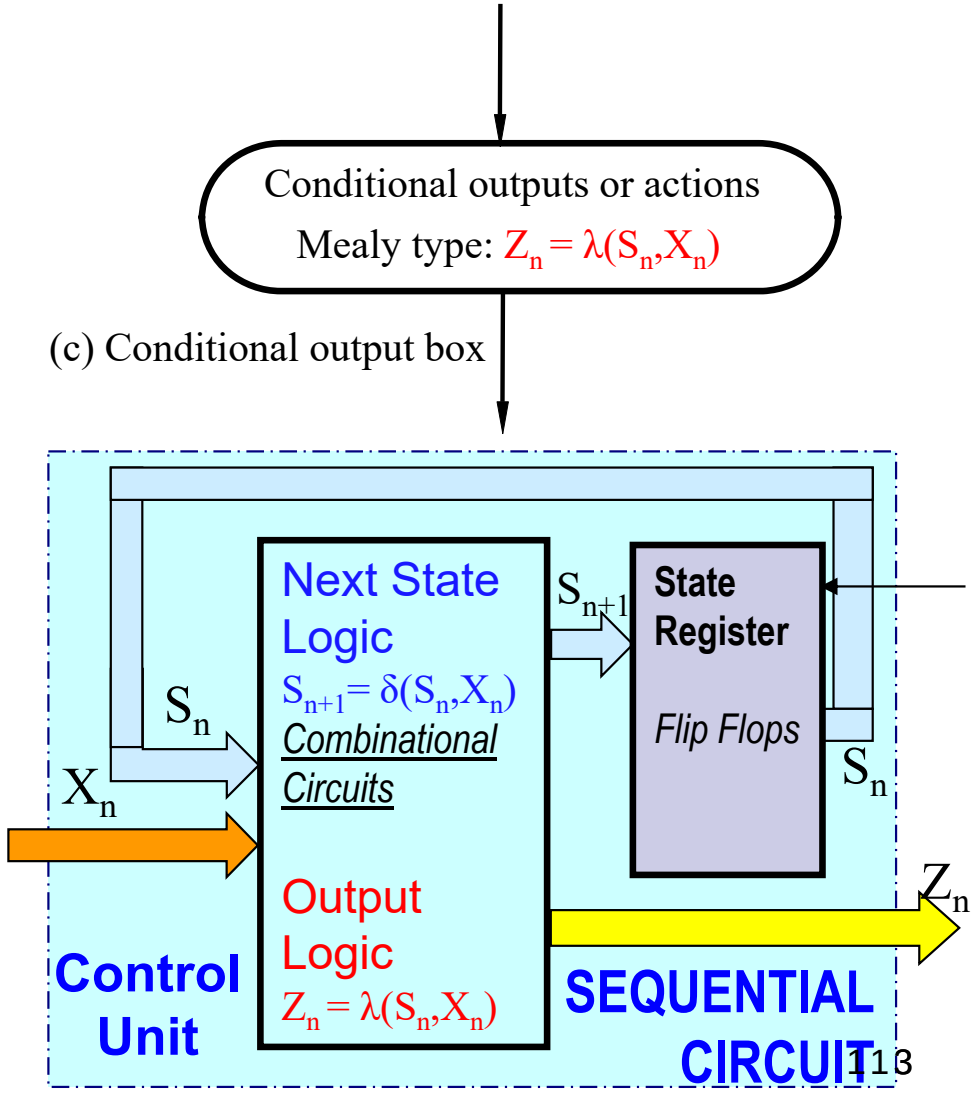
ASM#4 **Detailed Chart**



112

Symbols used in **ASM#4 detailed charts**



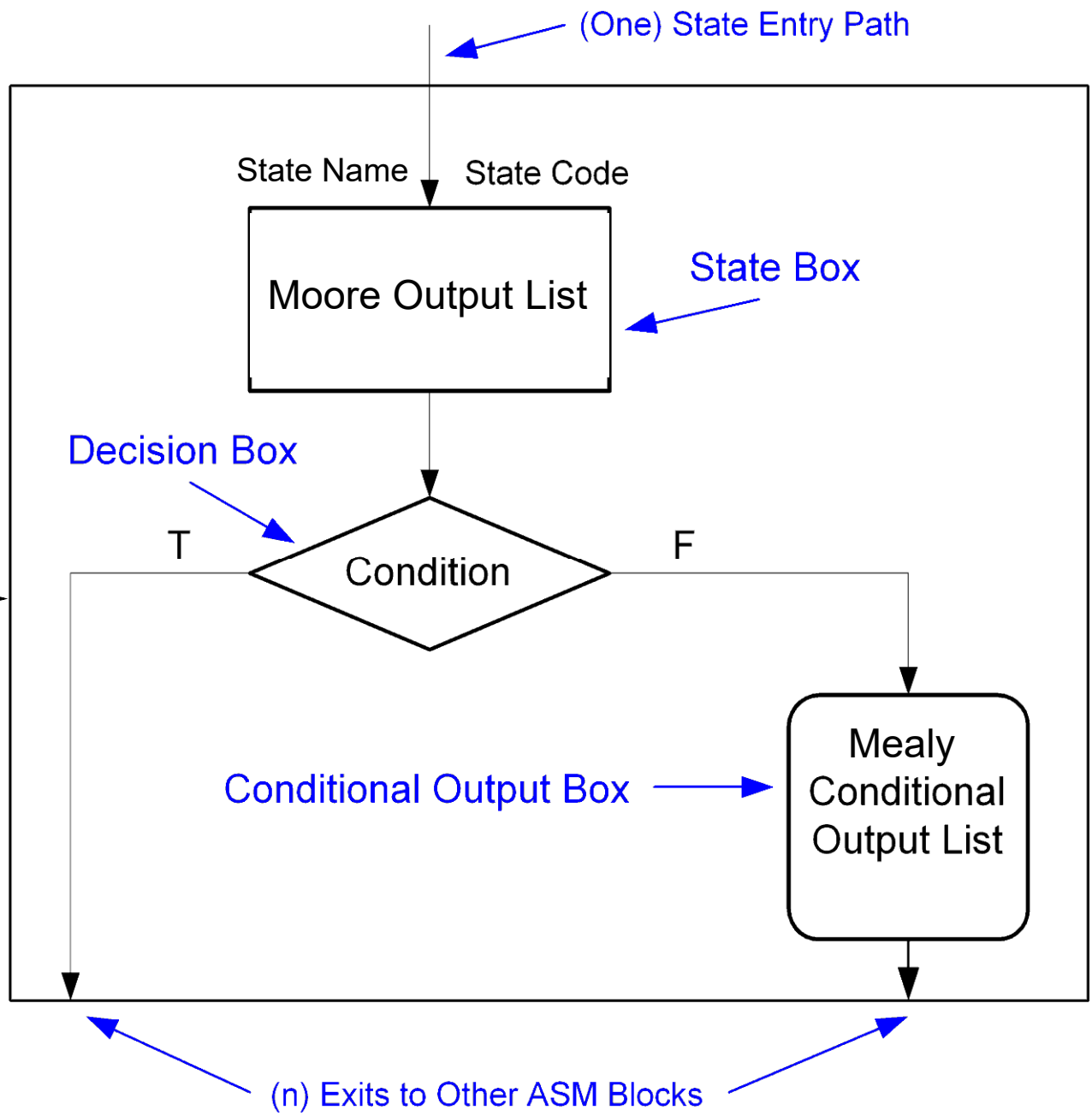


ASM Basic Block

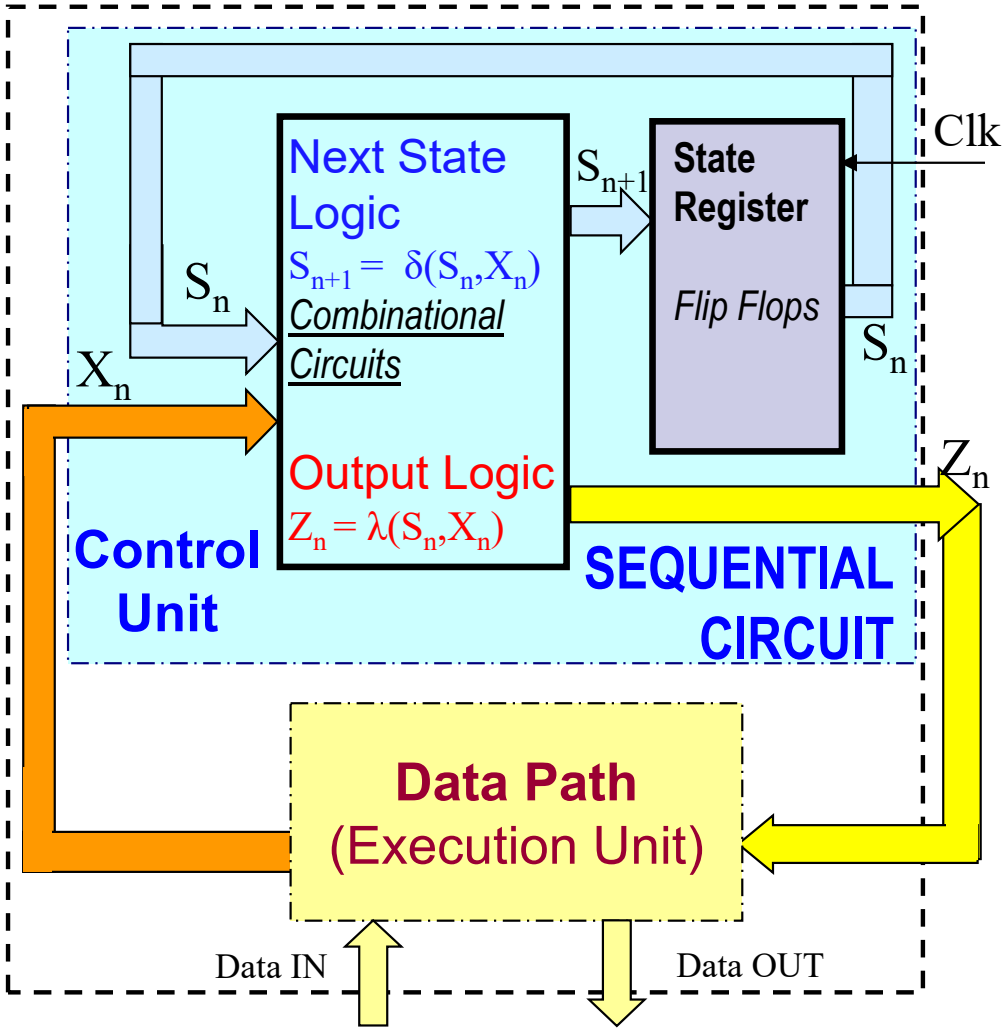
Recall the main steps in the ASM methodology:

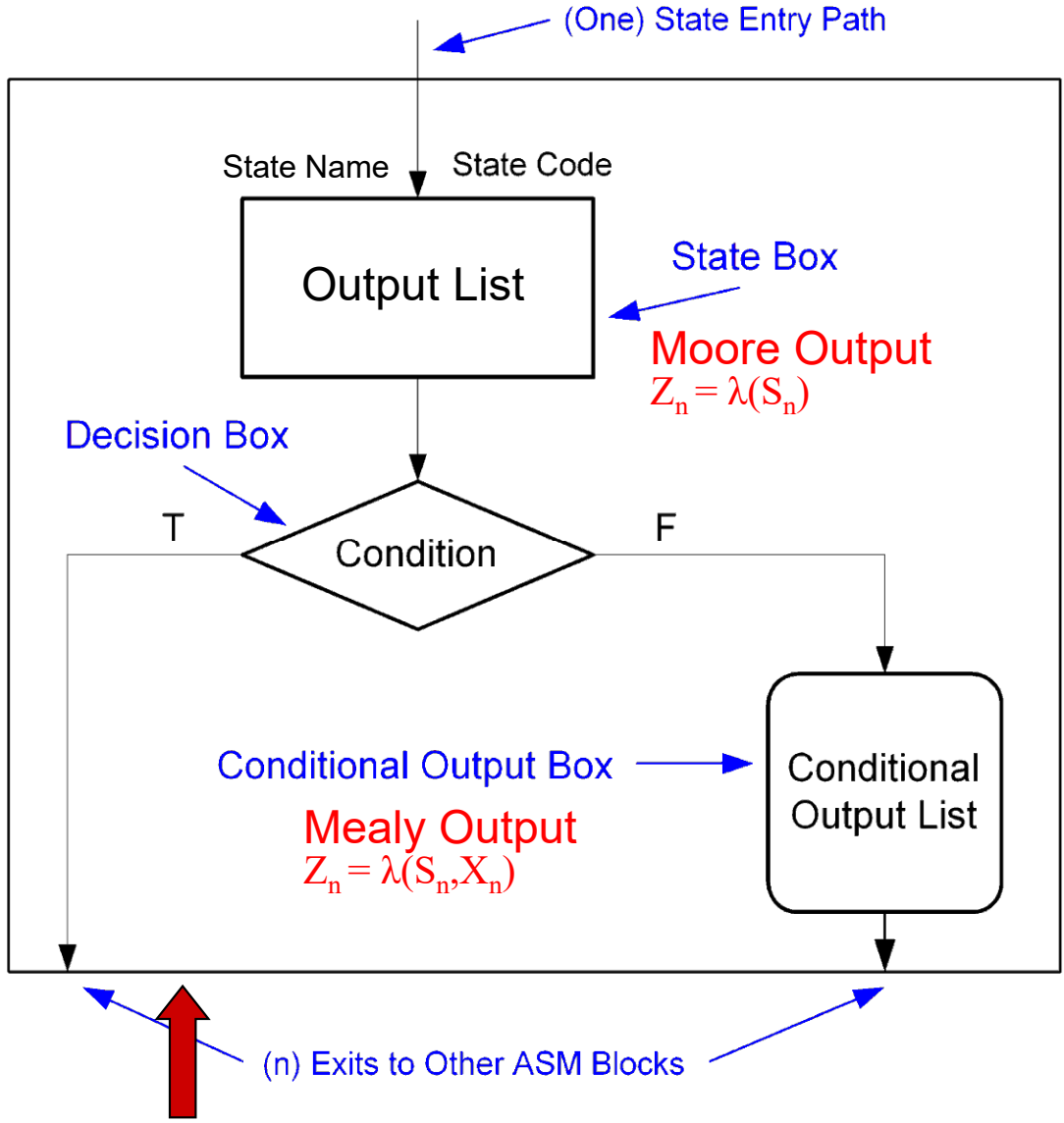
- Create an algorithm, using *pseudocode*, to describe the desired operation of the device
- Convert the pseudocode into an *ASM chart* (the basic brick = **ASM block**)
- Design the *datapath* based on the ASM chart
- Create a *detailed ASM chart* based on the datapath
- Design the *control logic* based on the detailed ASM chart

Combination of **datapath** and **control logic** makes up the actual logic system that will solve the original problem.



ASM#4 Detailed Chart



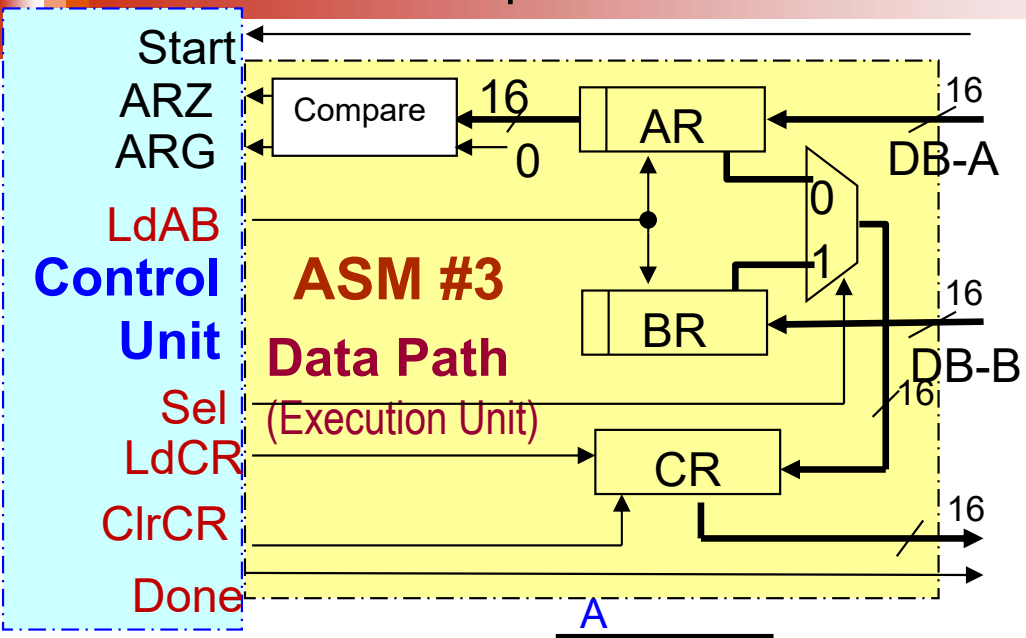


The basic brick = **ASM block** Combination of **datapath** and **control unit** makes up the actual logic system.

ASM#4 Detailed Chart

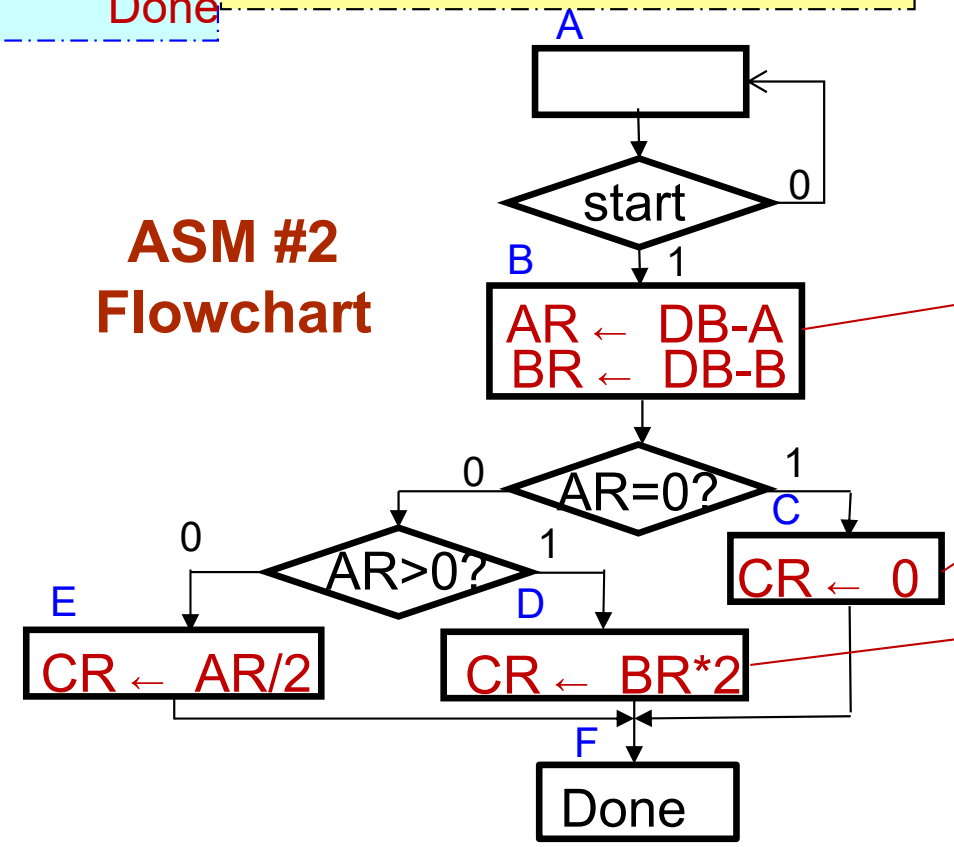
- There are some rules that have to be abided by during detailed ASM chart design
  1. For each *box* in the ASM flowchart, there should be a corresponding *state box* in the *detailed ASM chart*
  2. For each RTL operation in a state box of an ASM chart, the corresponding state box of the detailed ASM chart should contain an *output list* of control signals that are applied to the datapath in order to accomplish the RTL micro-operation described in the ASM chart. These control signals are generated by the *control unit* (CU) through its output function $Z = \lambda(S,X)$ - as CU is a sequential circuit.
     - All control signals are assumed to have the value '1' if specified and the value '0' if left unspecified.
  3. For each condition box in the ASM chart, there should be a corresponding condition box in the detailed ASM chart
     - Each condition box in the detailed ASM chart must contain a logical combination of *status signals* (coming from the datapath) that implements the combinational logic query in the corresponding ASM chart condition box
- The above rules provide the mapping between the ASM chart and the detailed ASM chart
- Pay special attention to the state box and conditional box mappings!
- Control logic is the heart of the digital circuit, and is often the most complex and detailed block of the entire system, and thus must be designed **very carefully!**
- The detailed ASM chart should easily be converted to a control path, as long as the aforementioned rules are followed

ASM #4 Detailed chart

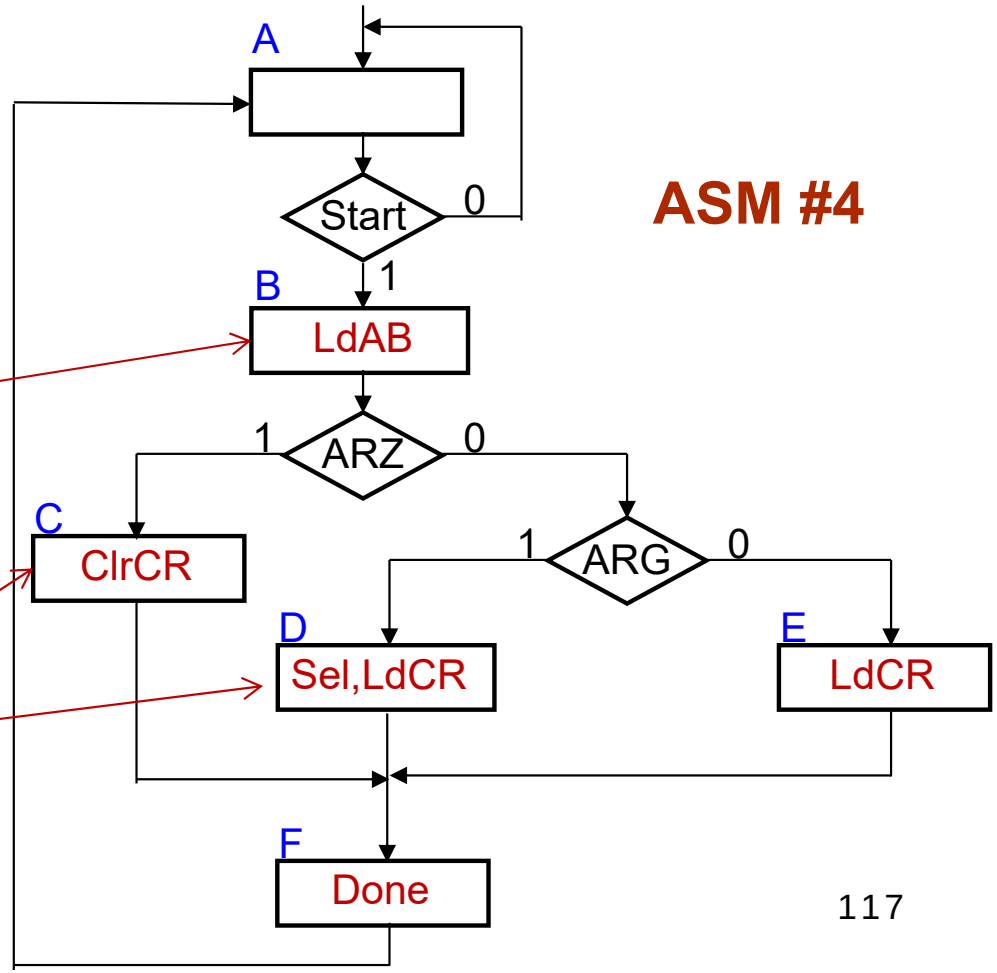


CU inputs = {Start, ARZ, ARG} = **Datapath** status outputs
CU outputs = {LdAB, Sel, LdCR, ClrCR, Done} =
= Control signals for **Datapath**

ASM #2 Flowchart



ASM #4



Control Unit

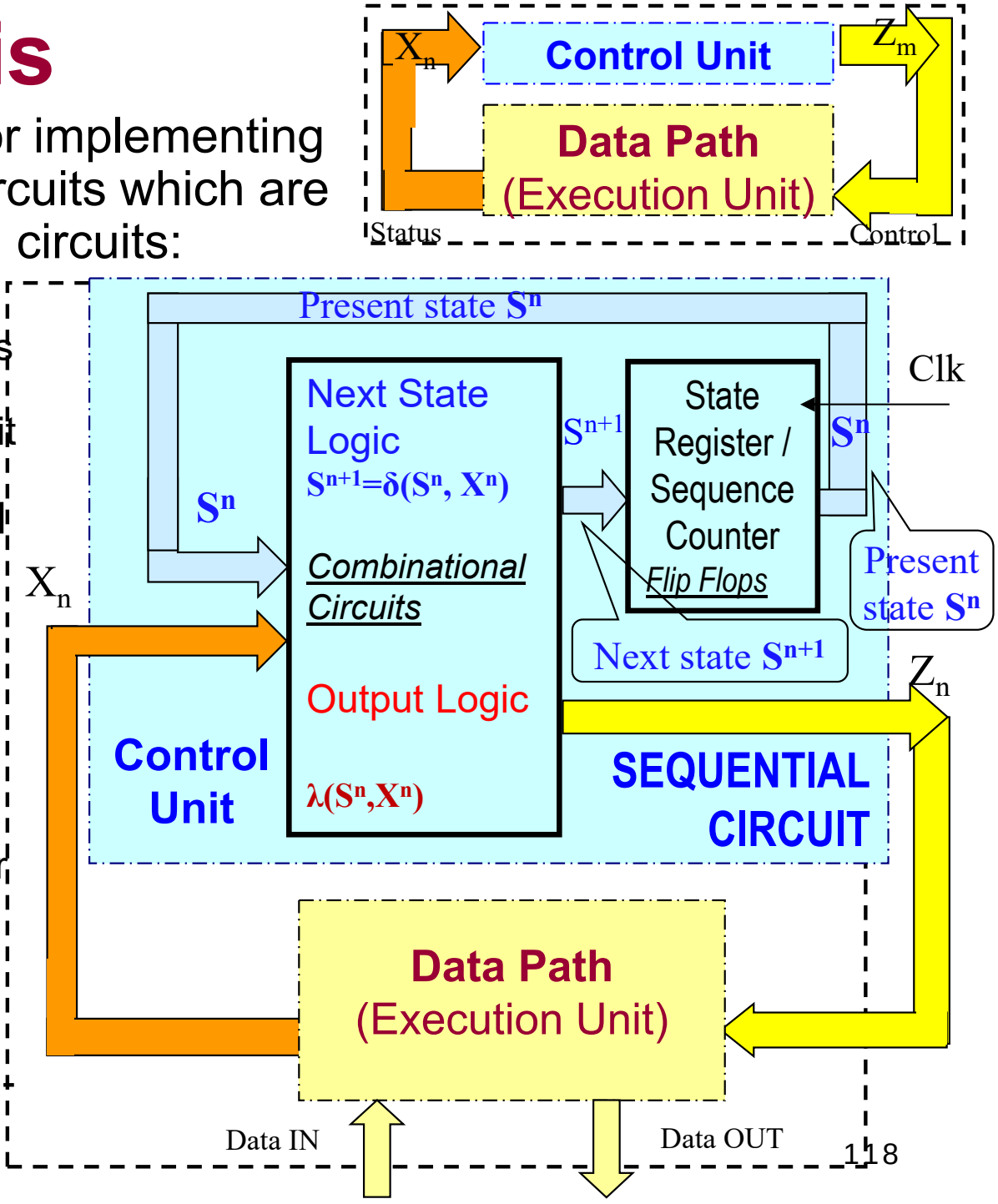
ASM#5 Synthesis

There are three major methods for implementing the State Register of sequential circuits which are used as **control logic** ASM-based circuits:

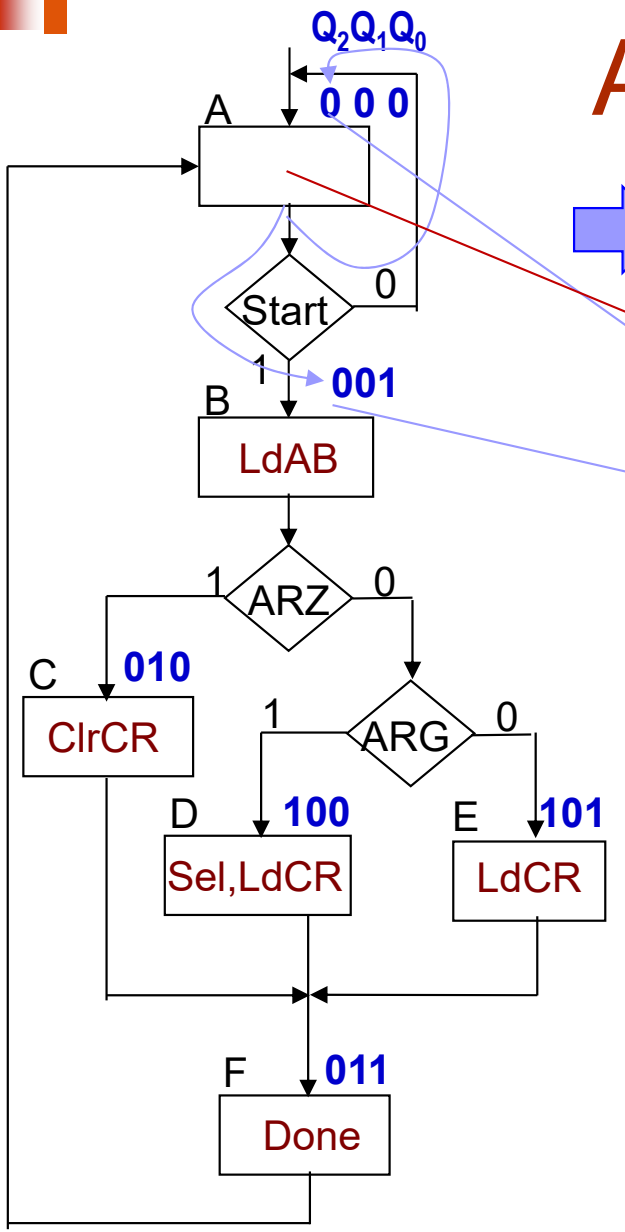
1. The **one-FF-per-state** method
   - Also called One-hot, we will see this method during this lecture
   - Most popular design technique, as it is simple and easy to model
2. The **Binary Encoded State** method
   - Uses state encoding to reduce the number of D flip-flops required in the state register
   - Control logic is spread out over several different digital devices
3. The **sequence-counter** method
   - Matches the cyclical nature of instruction execution on a computer
   - Control is simplified because of the cyclical state transition pattern

Control Units can be implemented as

- **Hardwired CU** - with gates and other MSI components (MUX, Decoders, multi-function registers) – Chapter 5
- **Microprogram CU** - ROM based.- Ch 7



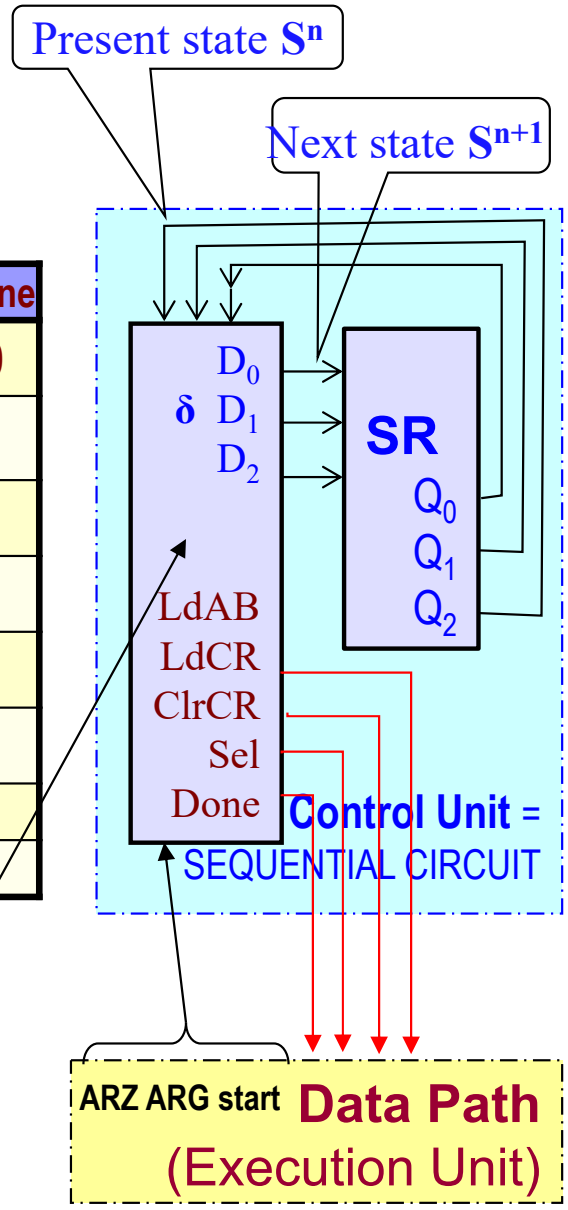
ASM #5.2a Synthesis with Gates



Derive from the ASM chart the transition table with MEV = map entered variable

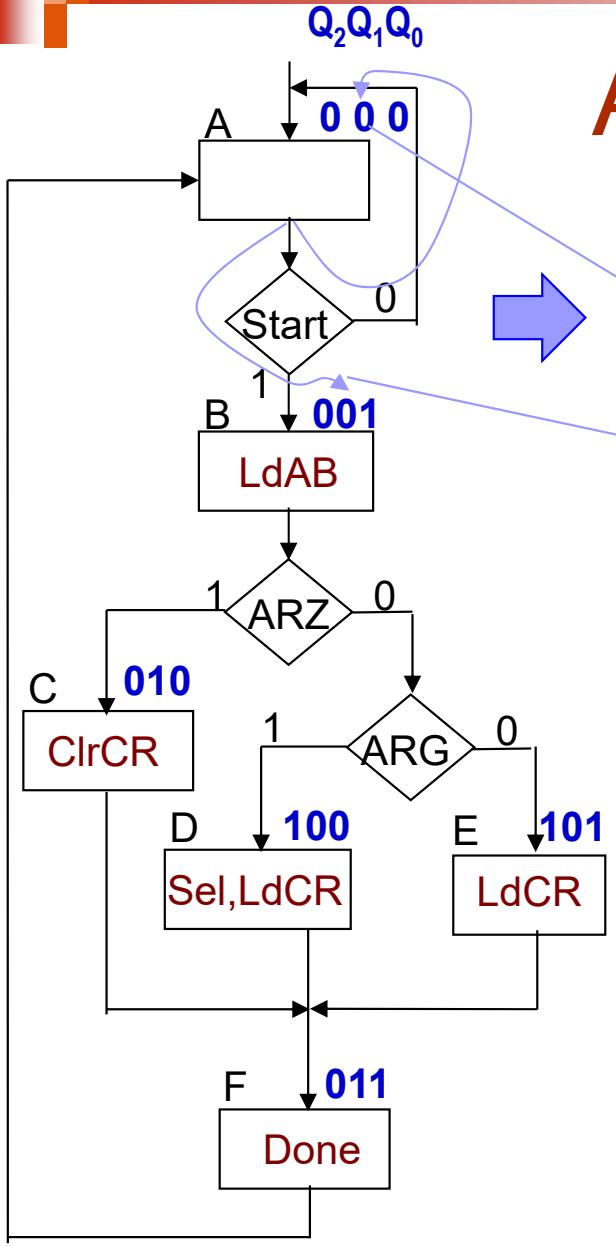
$S^{n+1}=\delta(S^n, X^n)$ $\lambda(S^n,X^n)$

State	Q2	Q1	Q0	$Q2^+$	$Q1^+$	$Q0^+$	LdAB	LdCR	ClrCR	Sel	Done
A	0	0	0	0			0	0	0	0	0
B	0	0	1								
C	0	1	0								
D	1	0	0								
E	1	0	1								
F	0	1	1								
	1	1	0								
	1	1	1								



δ and λ equations implemented with **gates**

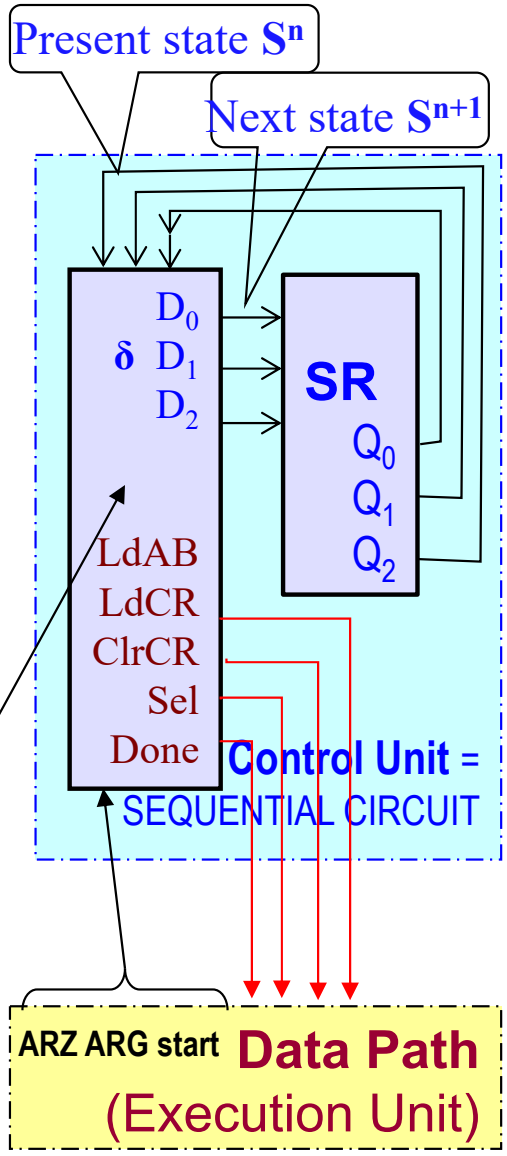
ASM #5.2a Synthesis with Gates



Derive transition table with MEV = map entered variable

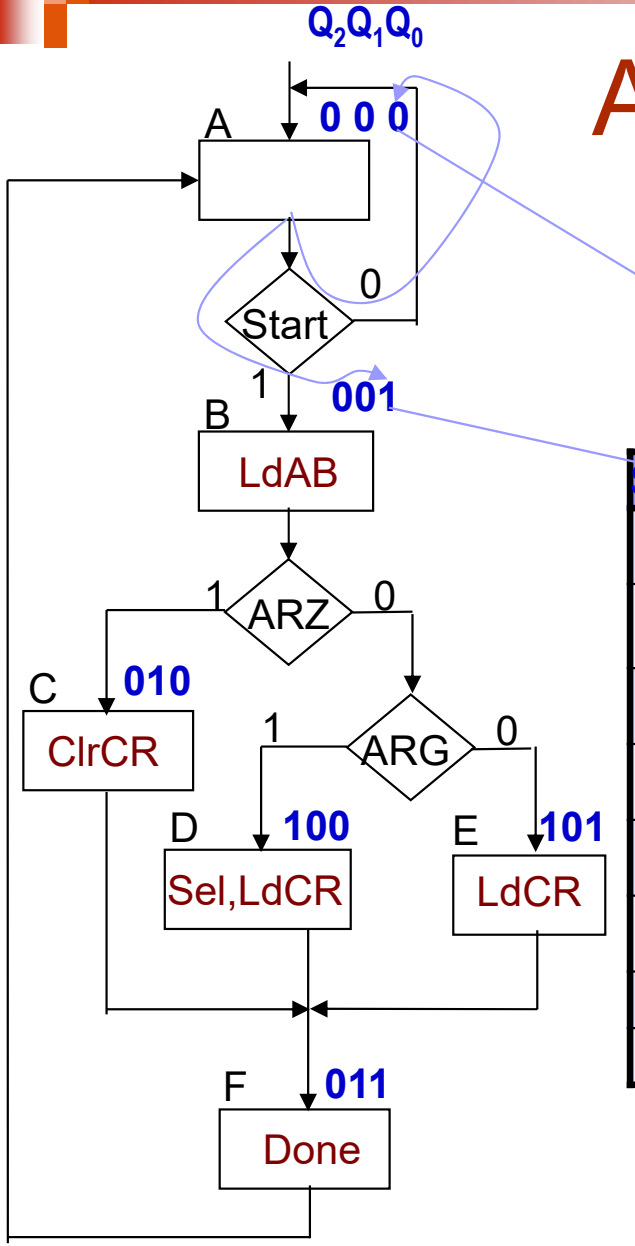
$S^{n+1}=\delta(S^n, X^n)$ $\lambda(S^n,X^n)$

State	Q2	Q1	Q0	Q2+	Q1+	Q0+	LdAB	LdCR	ClrCR	Sel	Done
A	0	0	0	0	0		0	0	0	0	0
B	0	0	1								
C	0	1	0								
D	1	0	0								
E	1	0	1								
F	0	1	1								
	1	1	0								
	1	1	1								



δ and λ equations implemented with **gates**

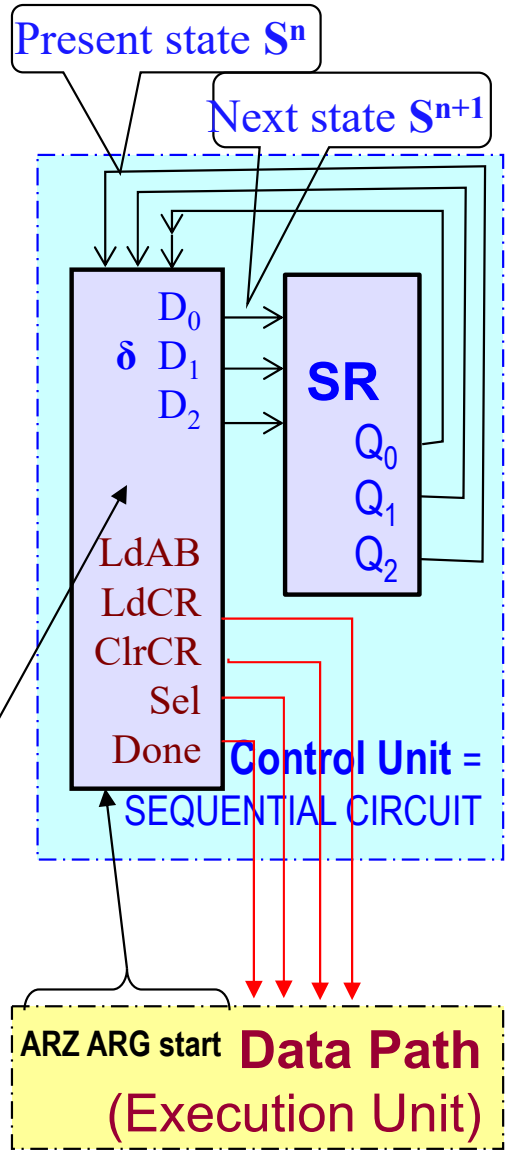
ASM #5.2a Synthesis with Gates



Derive transition table with MEV = map entered variable

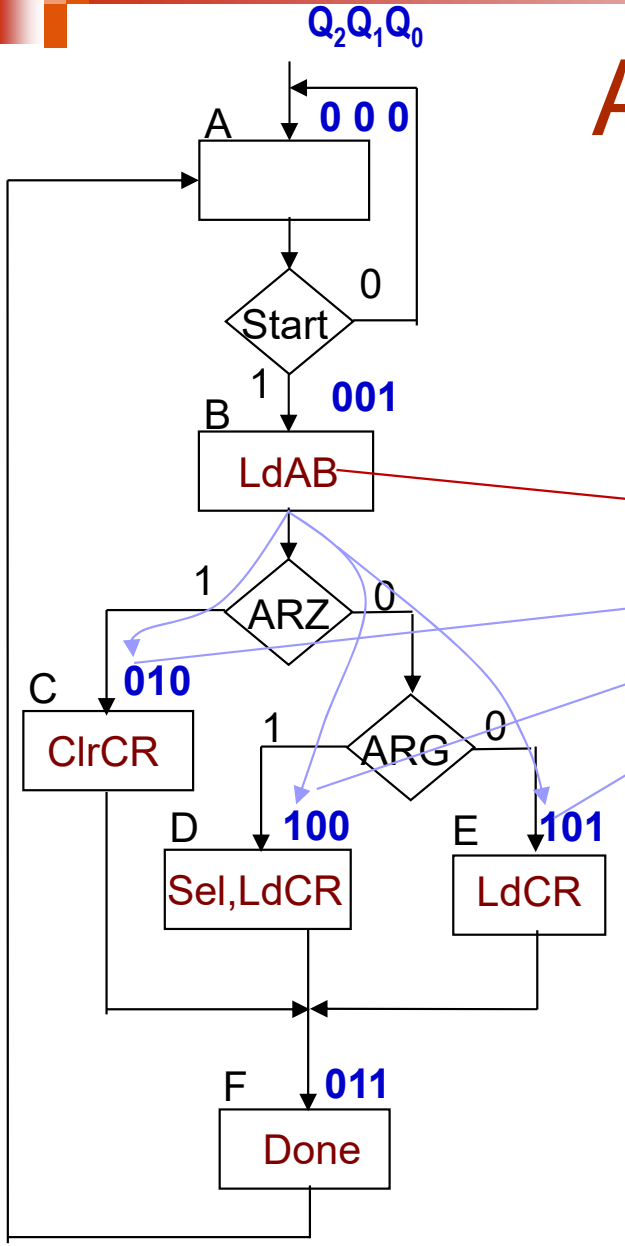
$S^{n+1}=\delta(S^n, X^n)$ $\lambda(S^n,X^n)$

State	Q2	Q1	Q0	Q2+	Q1+	Q0+	LdAB	LdCR	ClrCR	Sel	Done
A	0	0	0	0	0	Start	0	0	0	0	0
B	0	0	1								
C	0	1	0								
D	1	0	0								
E	1	0	1								
F	0	1	1								
	1	1	0								
	1	1	1								



δ and λ equations implemented with **gates**

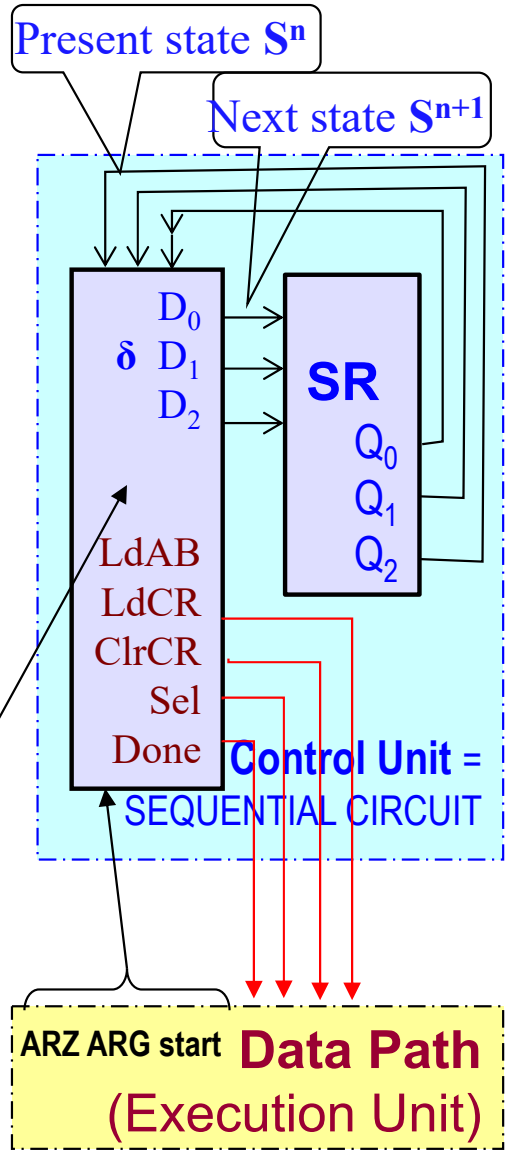
ASM #5.2a Synthesis with Gates



Derive transition table with
MEV = map entered variable

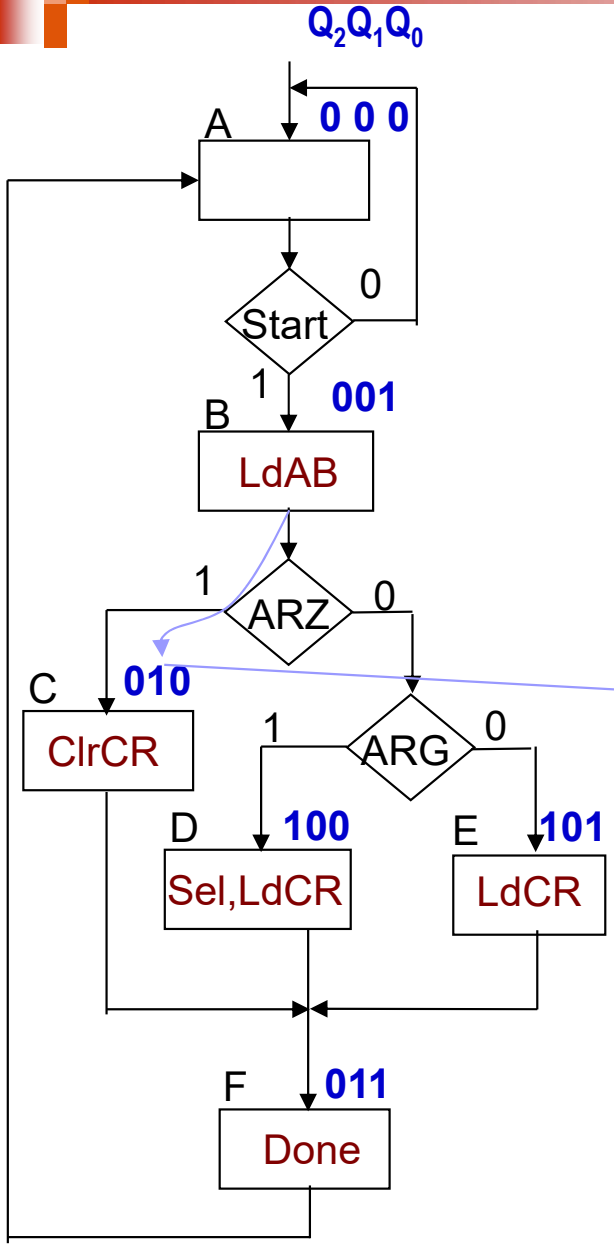
$S^{n+1}=\delta(S^n, X^n)$ $\lambda(S^n,X^n)$

State	Q2	Q1	Q0	Q2+	Q1+	Q0+	LdAB	LdCR	ClrCR	Sel	Done
A	0	0	0	0	0	Start	0	0	0	0	0
B	0	0	1	ARZ'			1	0	0	0	0
C	0	1	0								
D	1	0	0								
E	1	0	1								
F	0	1	1								
	1	1	0								
	1	1	1								



δ and λ equations implemented **with gates**

ASM #5.2a Synthesis with Gates

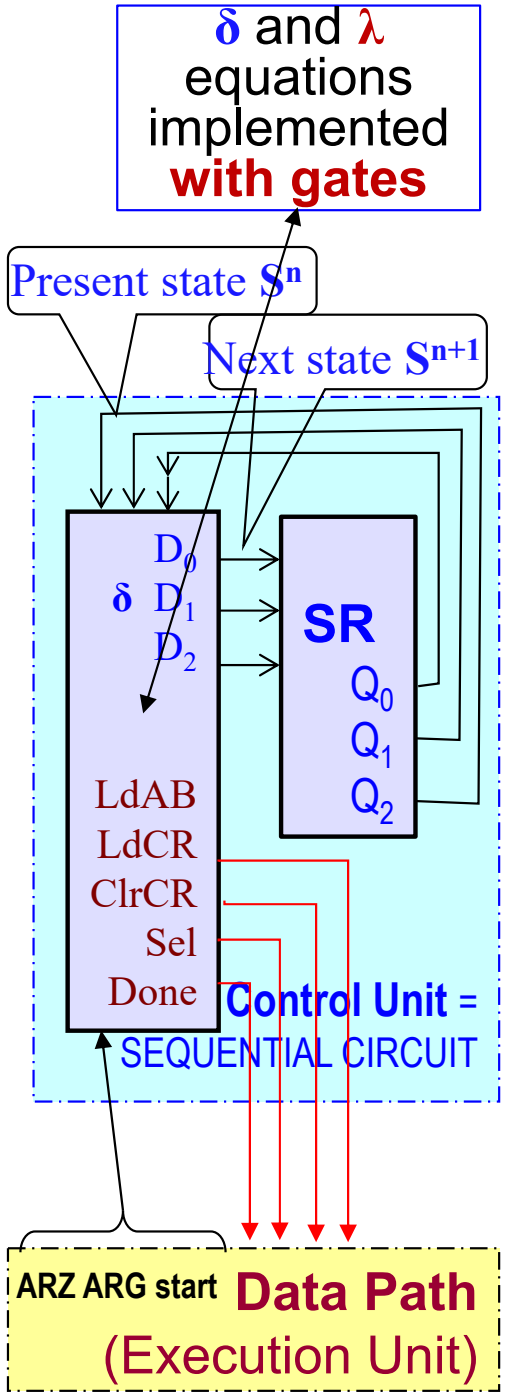


Derive transition table with MEV = map entered variable

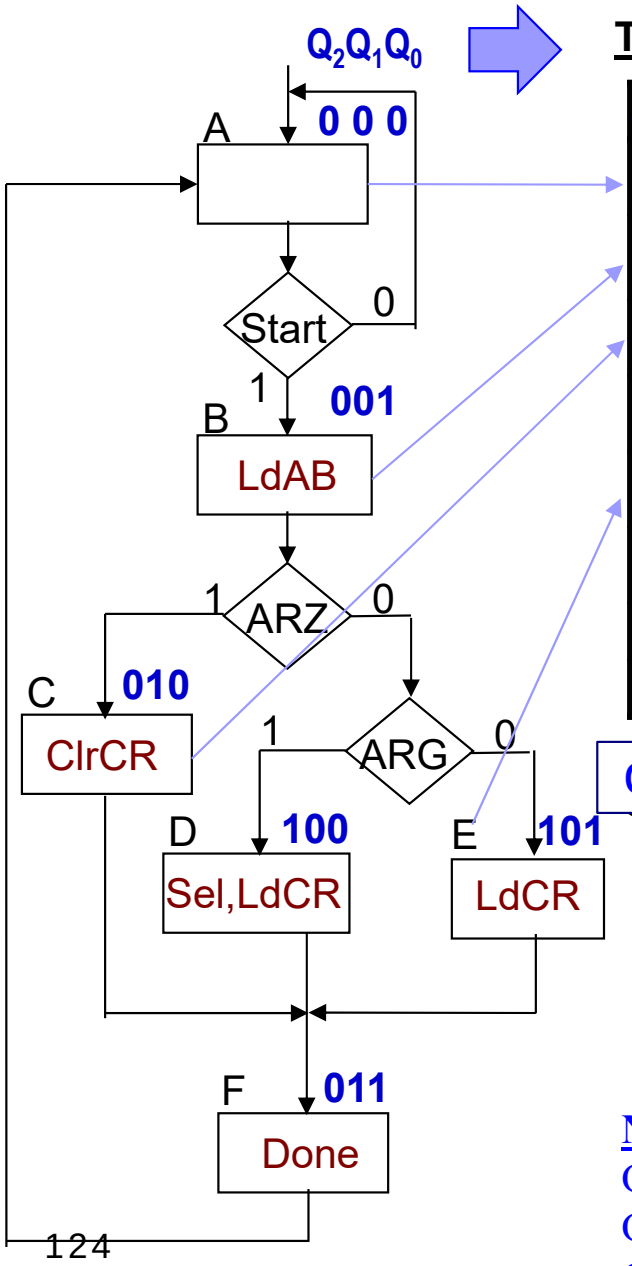
$S^{n+1}=\delta(S^n, X^n)$ $\lambda(S^n,X^n)$

State	Q2	Q1	Q0	Q2⁺	Q1⁺	Q0⁺	LdAB	LdCR	ClrCR	Sel	Done
A	0	0	0	0	0	Start	0	0	0	0	0
B	0	0	1	ARZ'	ARZ		1	0	0	0	0
C	0	1	0								
D	1	0	0								
E	1	0	1								
F	0	1	1								
	1	1	0								
	1	1	1								

Actually you should catch only 1's, i.e., write down in the transition table the MEV terms for which the FF's next state is 1… and don't bother about 0's …



ASM #5.2a Synthesis with Gates



Transition table with MEV

State	Q2	Q1	Q0	Q2⁺	Q1⁺	Q0⁺	LdAB	LdCR	ClrCR	Sel	Done
A	0	0	0	0	0	Start	0	0	0	0	0
B	0	0	1	ARZ'	ARZ	ARZ' ARG'	1	0	0	0	0
C	0	1	0	0	1	1	0	0	1	0	0
D	1	0	0	0	1	1	0	1	0	1	0
E	1	0	1	0	1	1	0	1	0	0	0
F	0	1	1	0	0	0	0	0	0	0	1
	1	1	0	x	x	x	x	x	x	x	x
	1	1	1	x	x	x	x	x	x	x	x

$Q0^+$

Q2 \ Q1Q0	00	01	11	10
0	0 Start	1 ARZ' ARG'	3 0	2 1
1	4 1	5 1	7 x	6 x

Output Logic $Z^n = \lambda(S^n)$

LdAB = Q2'Q1'Q0

LdCR =
= Q2'Q1Q0+Q2Q1'Q0'

ClrCR = Q2'Q1Q0'

Sel = Q2'Q1Q0

Done = Q2Q1Q0

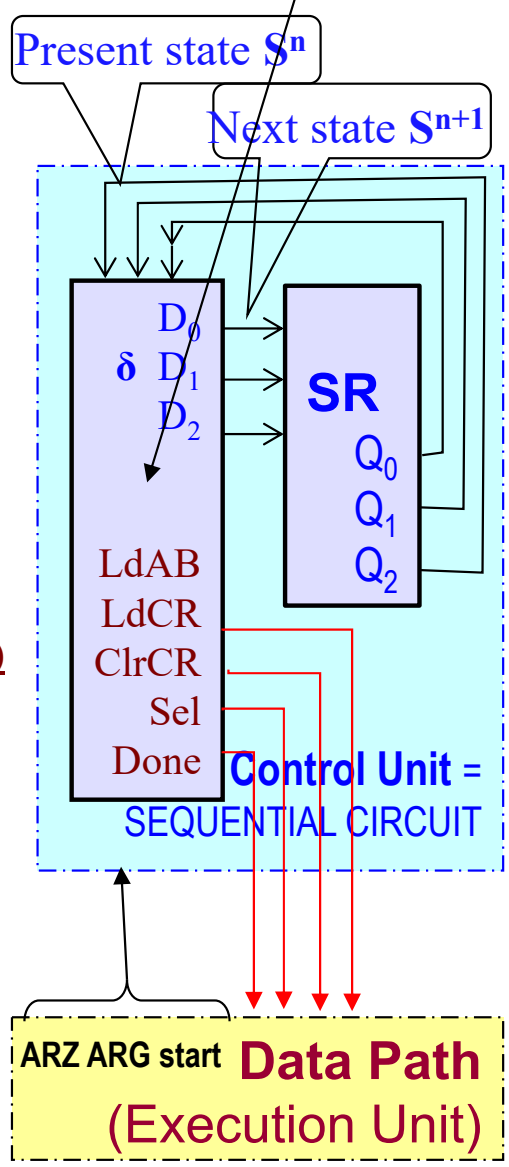
Next State Logic $S^{n+1}=\delta(S^n, X^n)$

$Q2^+ = Q2'Q1'Q0 \cdot ARZ'$

$Q1^+ = Q1'Q0 \cdot ARZ + Q2 + Q1'Q0$

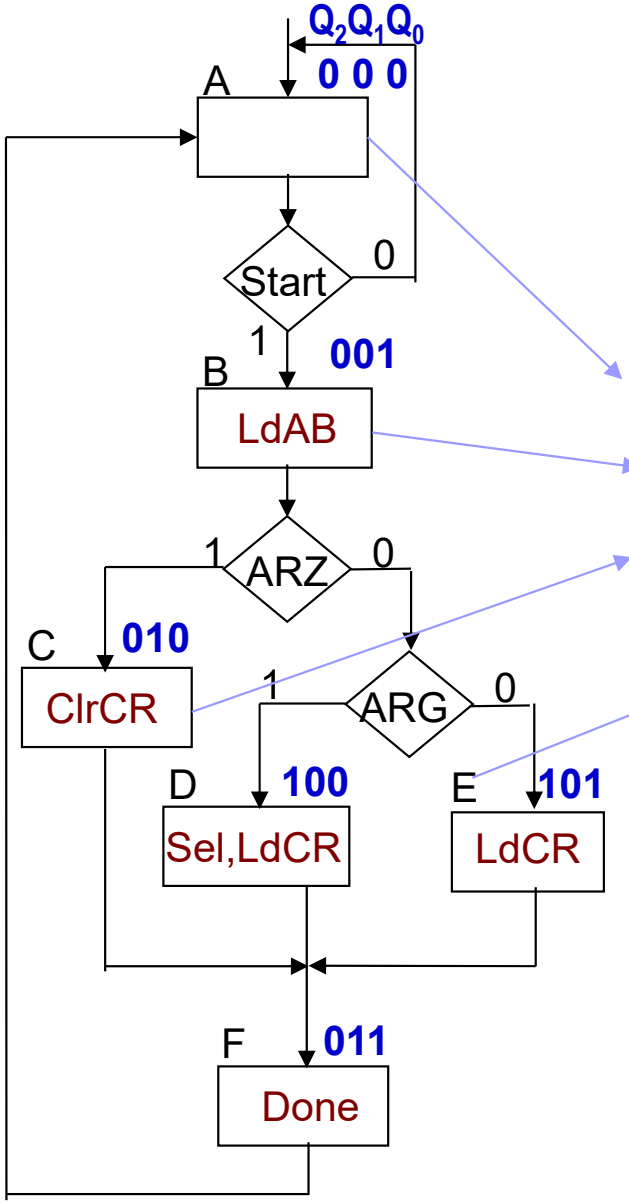
$Q0^+ = Q0'$ start $+ Q1'Q0 \cdot ARZ' \cdot ARG' + Q2 + Q1Q0'$

δ and λ equations implemented with gates



ASM #5.2b Synthesis with MUX's from transition table

δ and λ equations implemented with MUX's

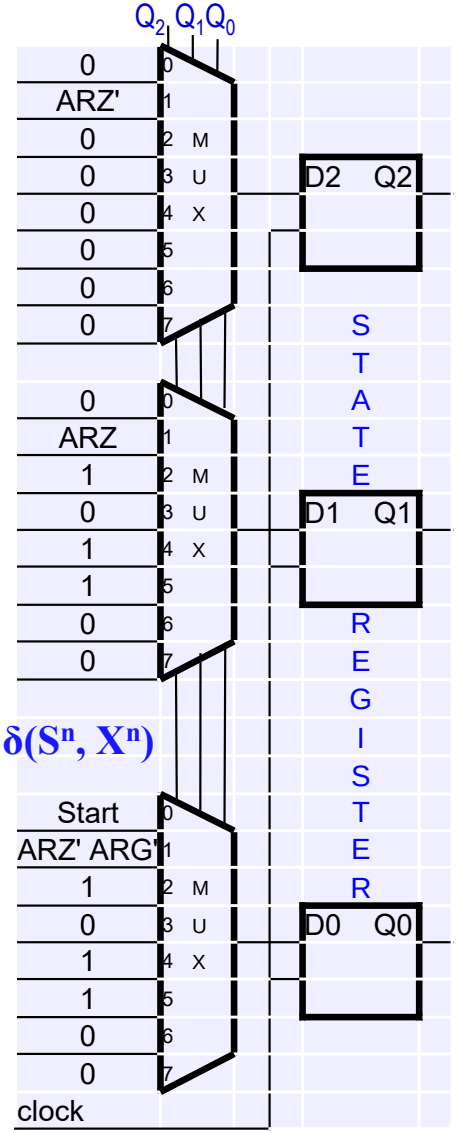


Transition table with MEV

$S^{n+1}=\delta(S^n, X^n)$ $Z^n=\lambda(S^n)$

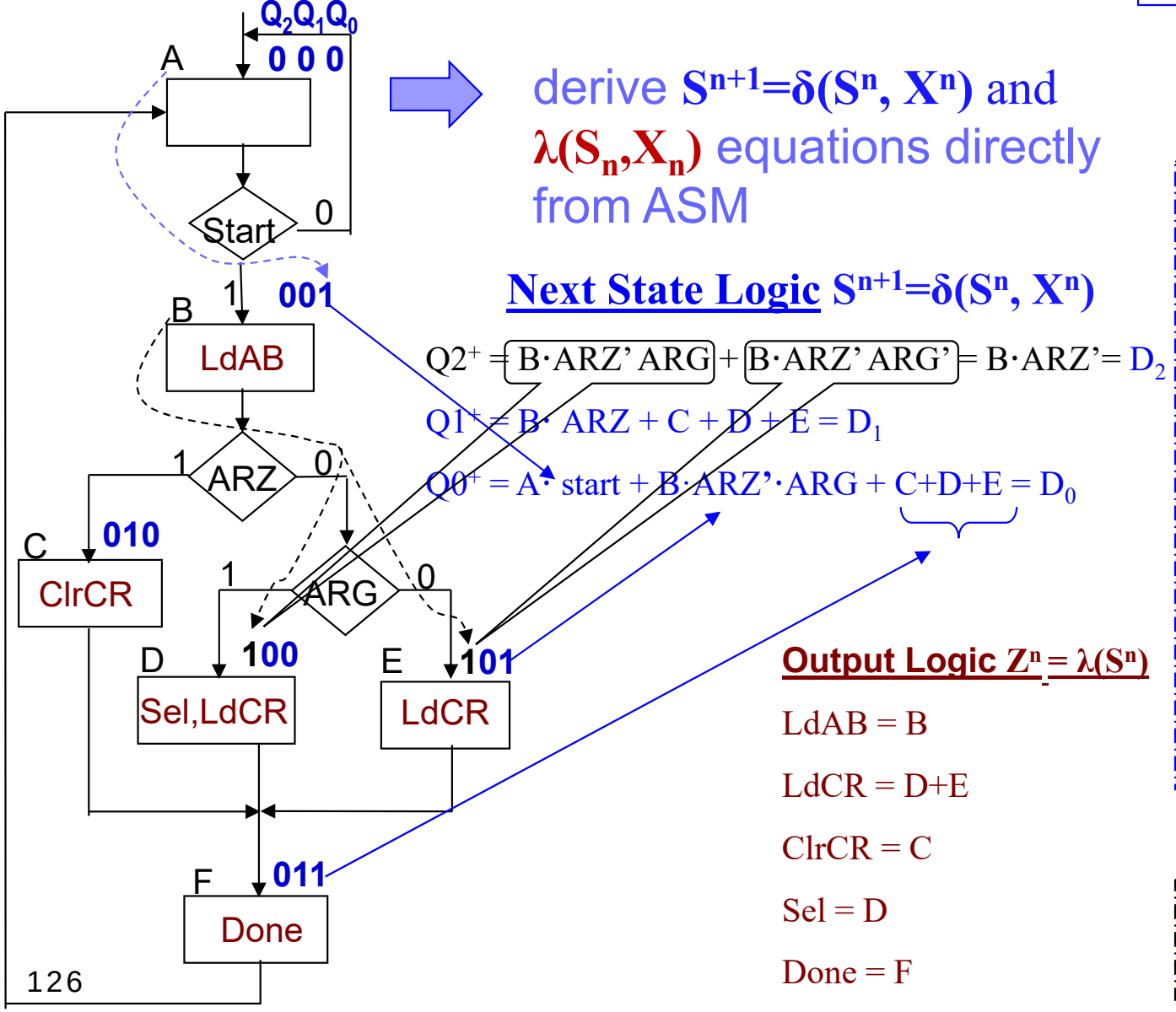
State	Q2	Q1	Q0	Q2⁺	Q1⁺	Q0⁺	LdAB	LdCR	ClrCR	Sel	Done
A	0	0	0	0	0	Start	0	0	0	0	0
B	0	0	1	ARZ'	ARZ	ARZ' ARG'	1	0	0	0	0
C	0	1	0	0	1	1	0	0	1	0	0
D	1	0	0	0	1	1	0	1	0	1	0
E	1	0	1	0	1	1	0	1	0	0	0
F	0	1	1	0	0	0	0	0	0	0	1
	1	1	0	x	x	x	x	x	x	x	x
	1	1	1	x	x	x	x	x	x	x	x

Use the transition table derived as before to implement δ with MUX's



ASM #5.2c Direct Synthesis with Gates and Decoder

δ and λ equations implemented with state **DECODER** (equivalent of ONE HOT encoding) & **gates**



derive $S^{n+1}=\delta(S^n, X^n)$ and $\lambda(S_n,X_n)$ equations directly from ASM

Next State Logic $S^{n+1}=\delta(S^n, X^n)$

$Q2^+ = B\cdot ARZ' ARG + B\cdot ARZ' ARG' = B\cdot ARZ' = D_2$

$Q1^+ = B\cdot ARZ + C + D + E = D_1$

$Q0^+ = A\cdot start + B\cdot ARZ'\cdot ARG + C+D+E = D_0$

Output Logic $Z^n = \lambda(S^n)$

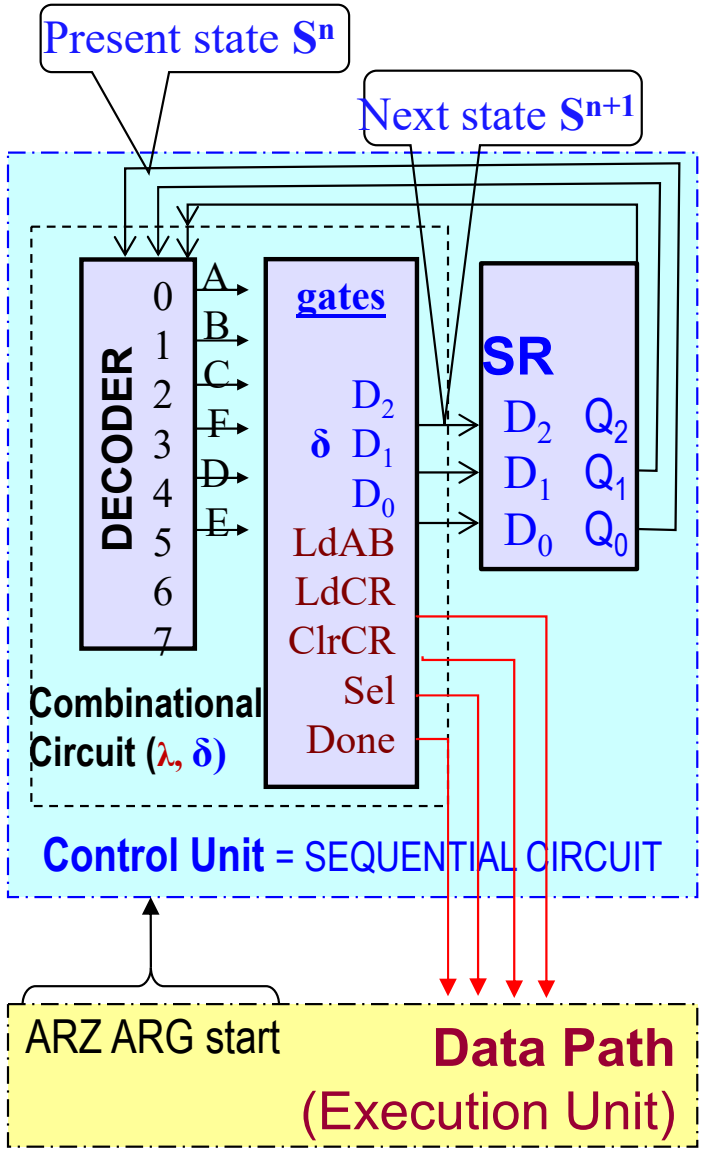
LdAB = B

LdCR = D+E

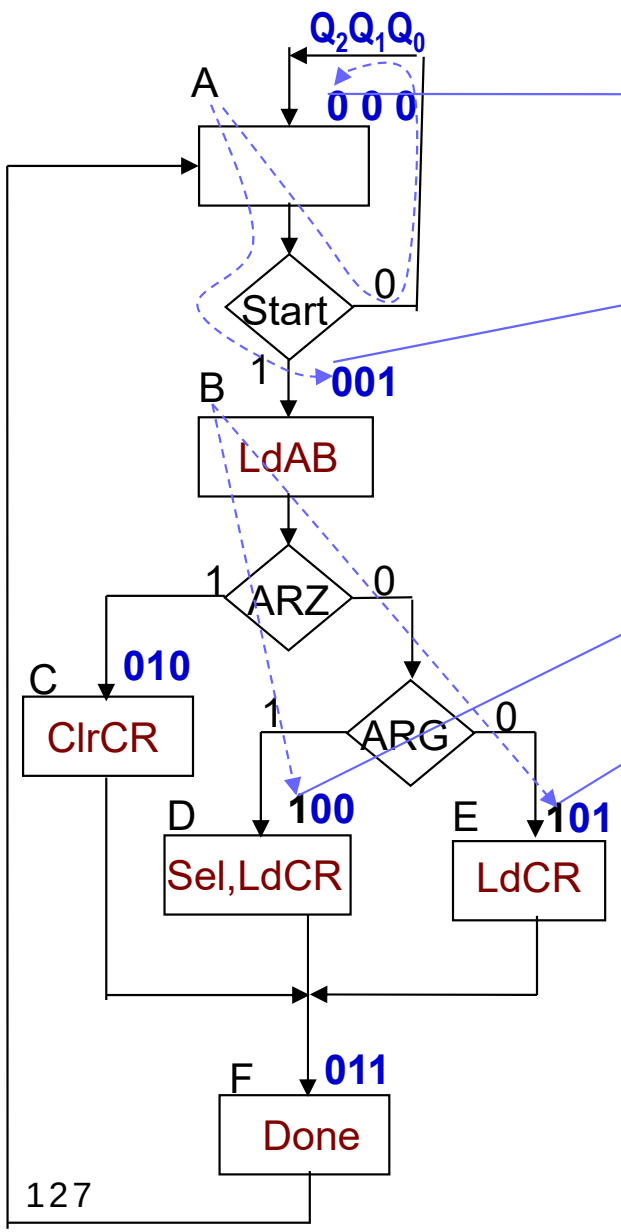
ClrCR = C

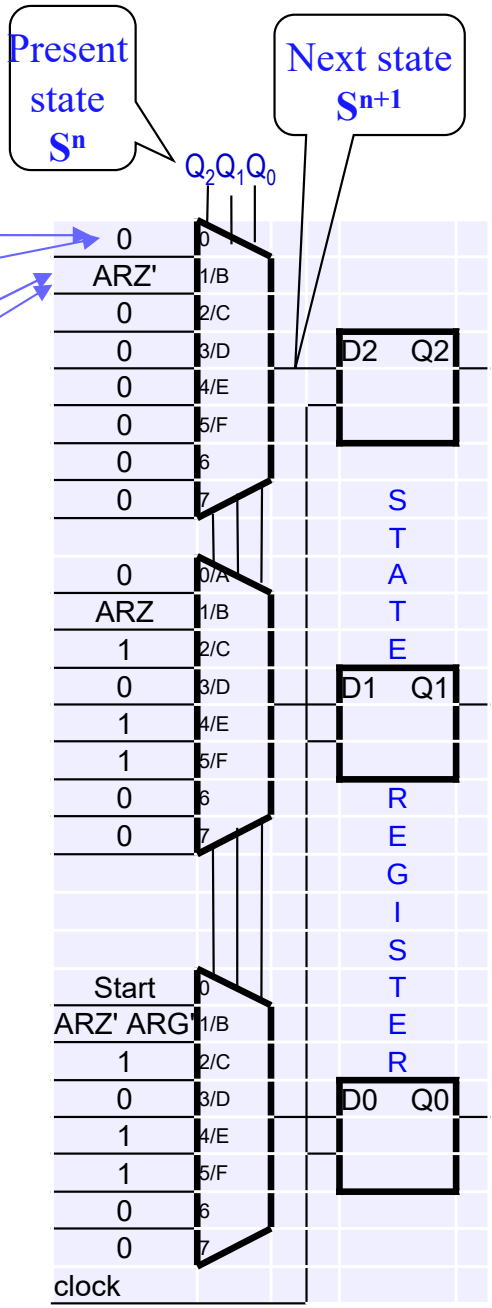
Sel = D

Done = F



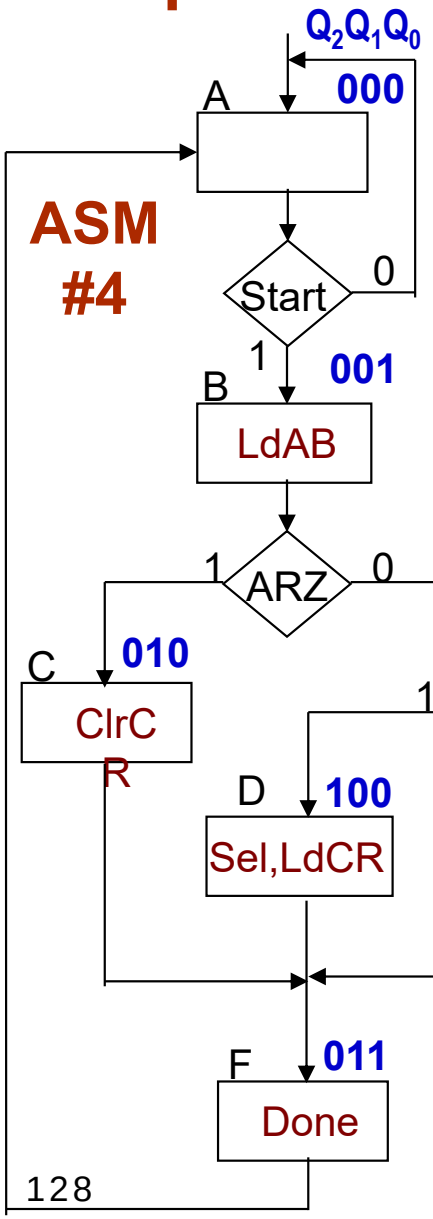
ASM #5.2d Direct Synthesis with MUX's





δ implemented with MUX's directly from ASM, without deriving the transition table or equations first

ASM #5.1 with Sequence Counter



ASM #4

State Register = Sequence Counter SC = multi-function register $Q_2Q_1Q_0$ with
- parallel load (LdSC),
- count up (SC++),
- clear (ClrSC)

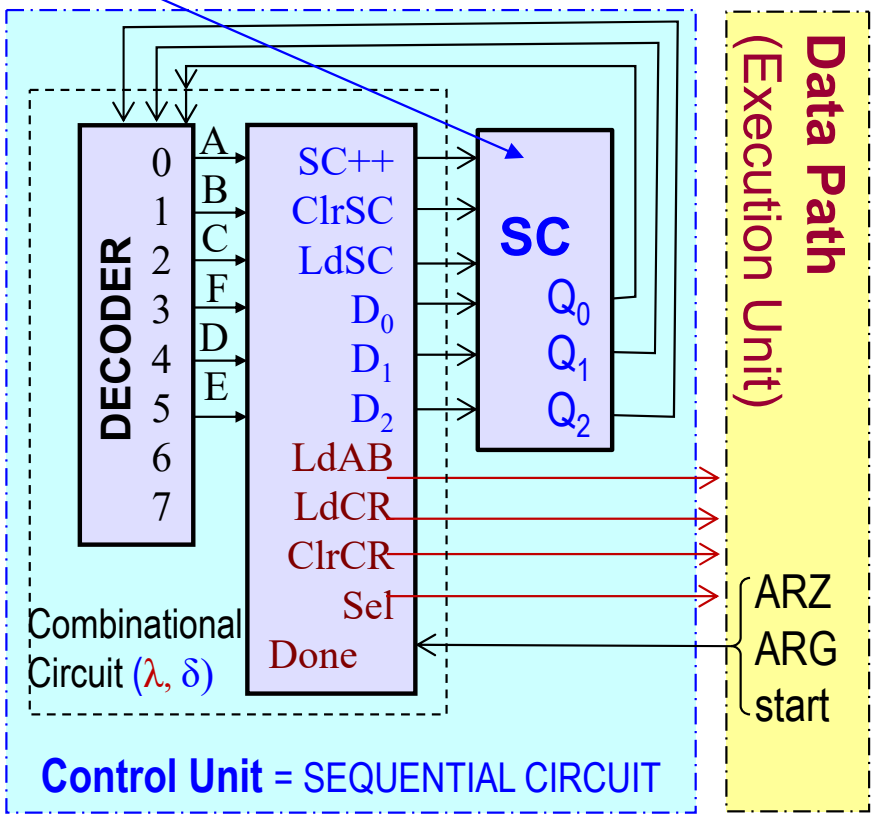
	ASM #2 / RTL		δ				λ				
	$Z^n = \lambda$	$S^{n+1} = \delta$	SC++	ClrSC	LdSC	SC^{n+1} $D_2D_1D_0$	LdAB	LdCR	ClrCR	Sel	Done
A Start:		SC←B	1								
B:	AR← DB-A, BR← DB-B						1				
B ARZ:		SC←C	1								
B·ARZ'·ARG:		SC←D			1	100					
B·ARZ'ARG':		SC←E			1	101					
C:	CR←0	SC←F	1						1		
D:	CR←BR*2	SC←F			1	011		1		1	
E:	CR←AR/2	SC←F			1	011		1			
F:		SC←A		1							1

The Next State Logic δ is ©ontrolled by

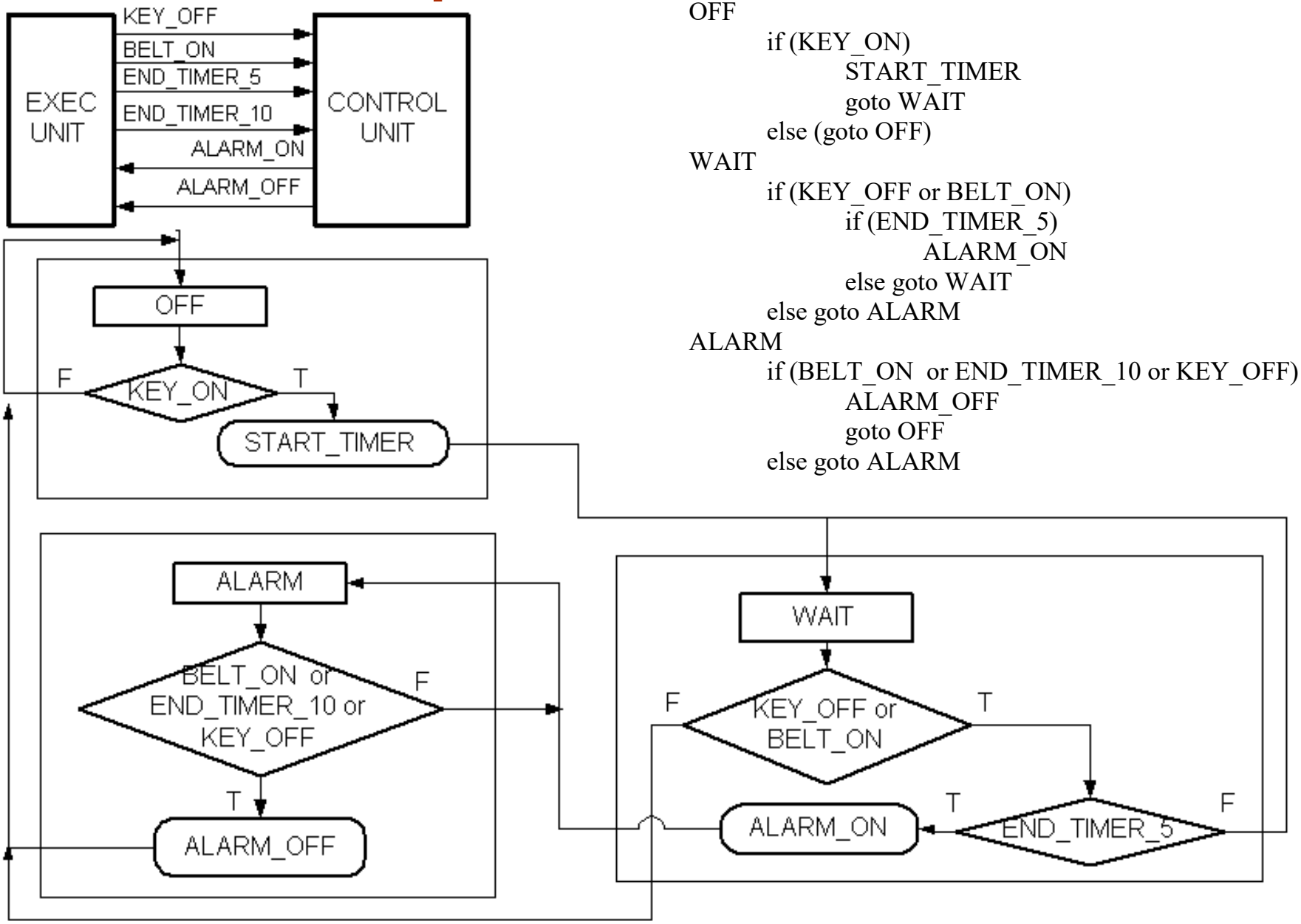
$SC{+}{+} = A{\cdot}start + B{\cdot}ARZ + C$
$ClrSC = F$
$LdSC = B{\cdot}ARZ'{\cdot}ARG + B{\cdot}ARZ'ARG' + D + E$
$D_0 = B{\cdot}ARZ'ARG' + D + E$
$D_1 = D + E$
$D_2 = B{\cdot}ARZ'{\cdot}ARG + B{\cdot}ARZ'ARG'$

Output Logic $Z^n = \lambda(S^n)$

$LdAB = B$
$LdCR = D + E$
$ClrCR = C$
$Sel = D$
$Done = F$



ASM Example




```
OFF
        if (KEY_ON)
                START_TIMER
                goto WAIT
        else (goto OFF)
WAIT
        if (KEY_OFF or BELT_ON)
                if (END_TIMER_5)
                        ALARM_ON
                else goto WAIT
        else goto ALARM
ALARM
        if (BELT_ON  or END_TIMER_10 or KEY_OFF)
                ALARM_OFF
                goto OFF
        else goto ALARM
```

Control Unit Design
Lab 4

Dr. Voicu Groza

SITE Hall, Room 5017
562 5800 ext. 2159
Groza@EECS.uOttawa.ca

uOttawa

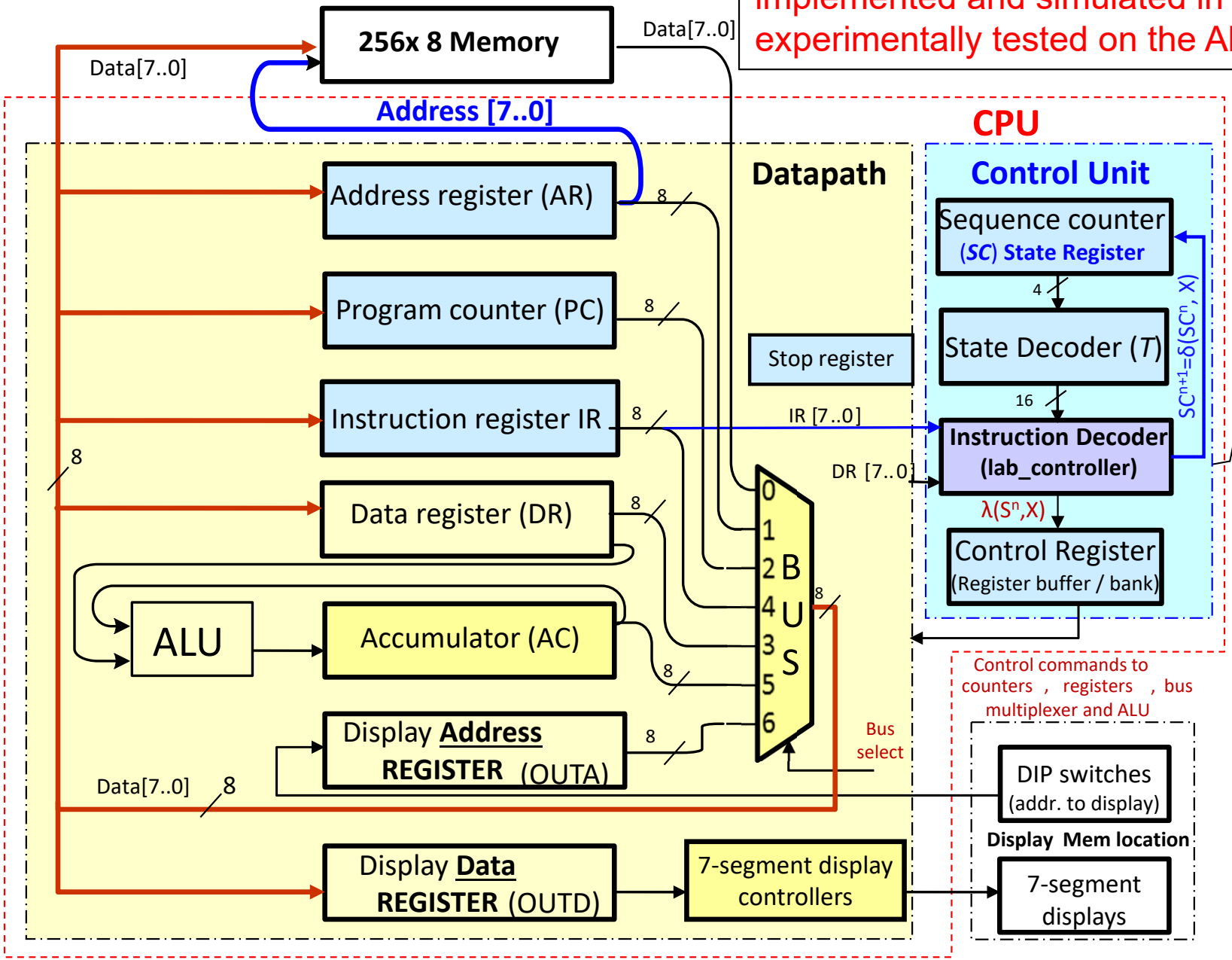
L'Université canadienne
Canada's university

Université d'Ottawa | University of Ottawa

www.uOttawa.ca

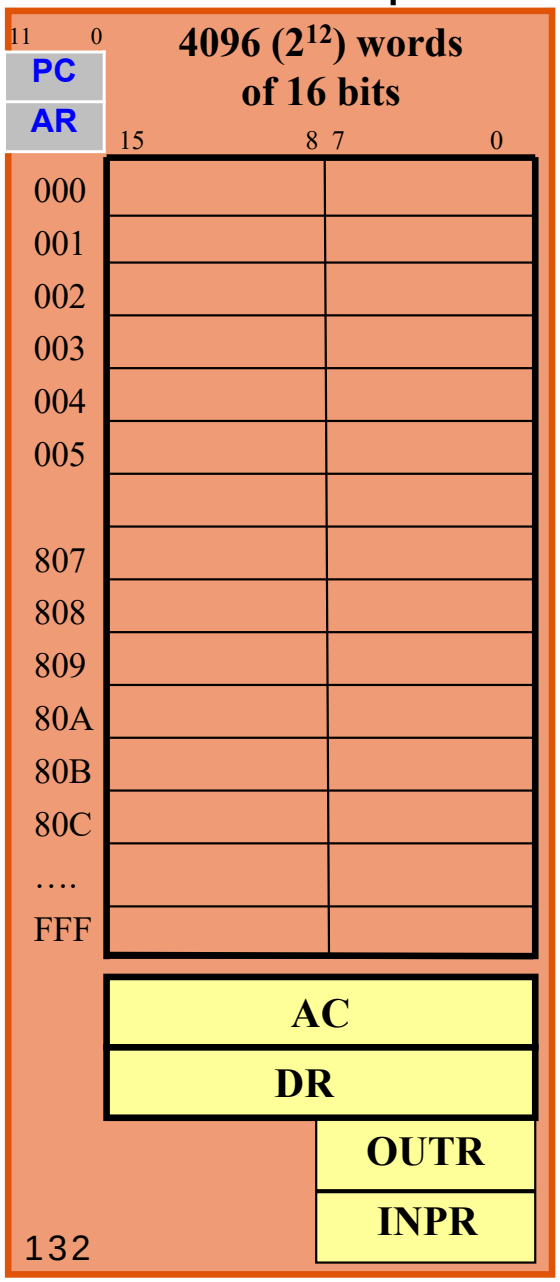
Lab#4 Computer Block Diagram

All modules are given in LAB #4, except this controller which has to be designed, implemented and simulated in Quartus and experimentally tested on the Altera platform

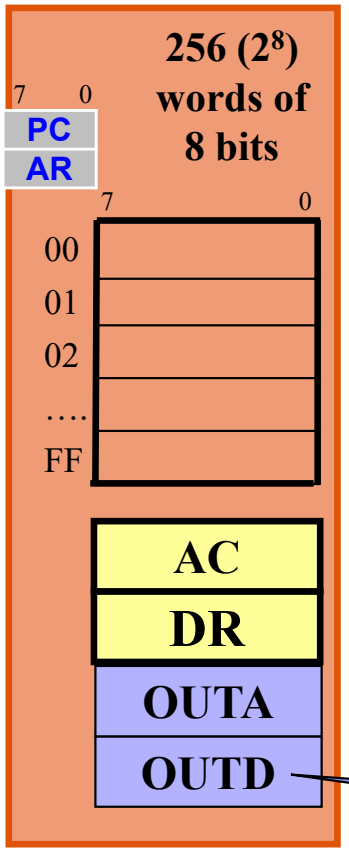


Programmer's View

Basic Computer

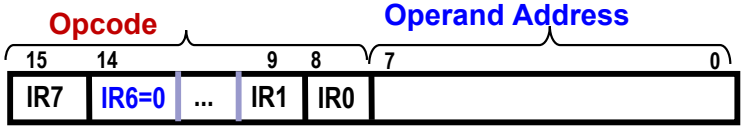


Lab Computer



Instructions Format

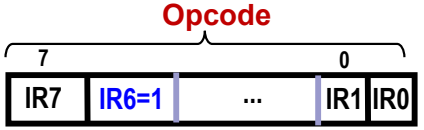
Memory reference (IR6 =0)

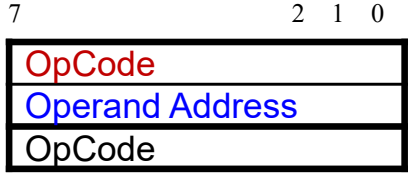


Direct Addressing (IR7 = 0)

Indirect Addressing (IR7 = 1)

Register reference (IR6 =1)





Memory Reference Instruction

Register Ref. Instrc.

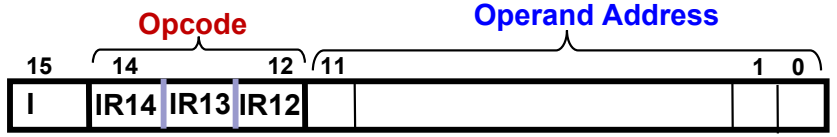
		IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
		I	Mem /Reg	O	p	C	o	d	e
Memory Reference	Direct Addr X_0	0	0	x	x	x	x	x	x
	Indirect Addr. X_2	1	0	x	x	x	x	x	x
Reg. Ref.	Instr. X_0	x	1	x	x	x	x	x	x

Visualize in OUTD the contents of the memory location from address OUTA

Memory-reference instruction format:

Direct Addressing (I =0)

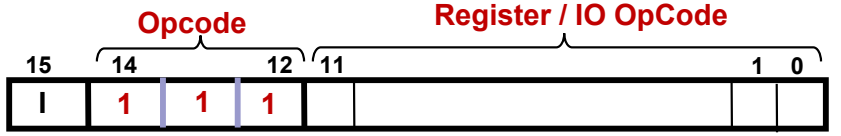
Indirect Addressing (I =1)



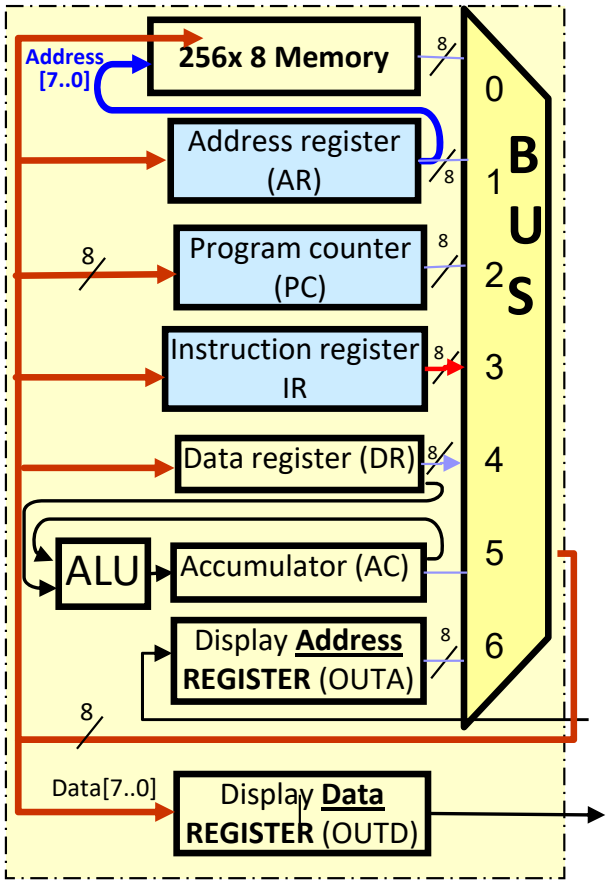
RR / IO instruction format:

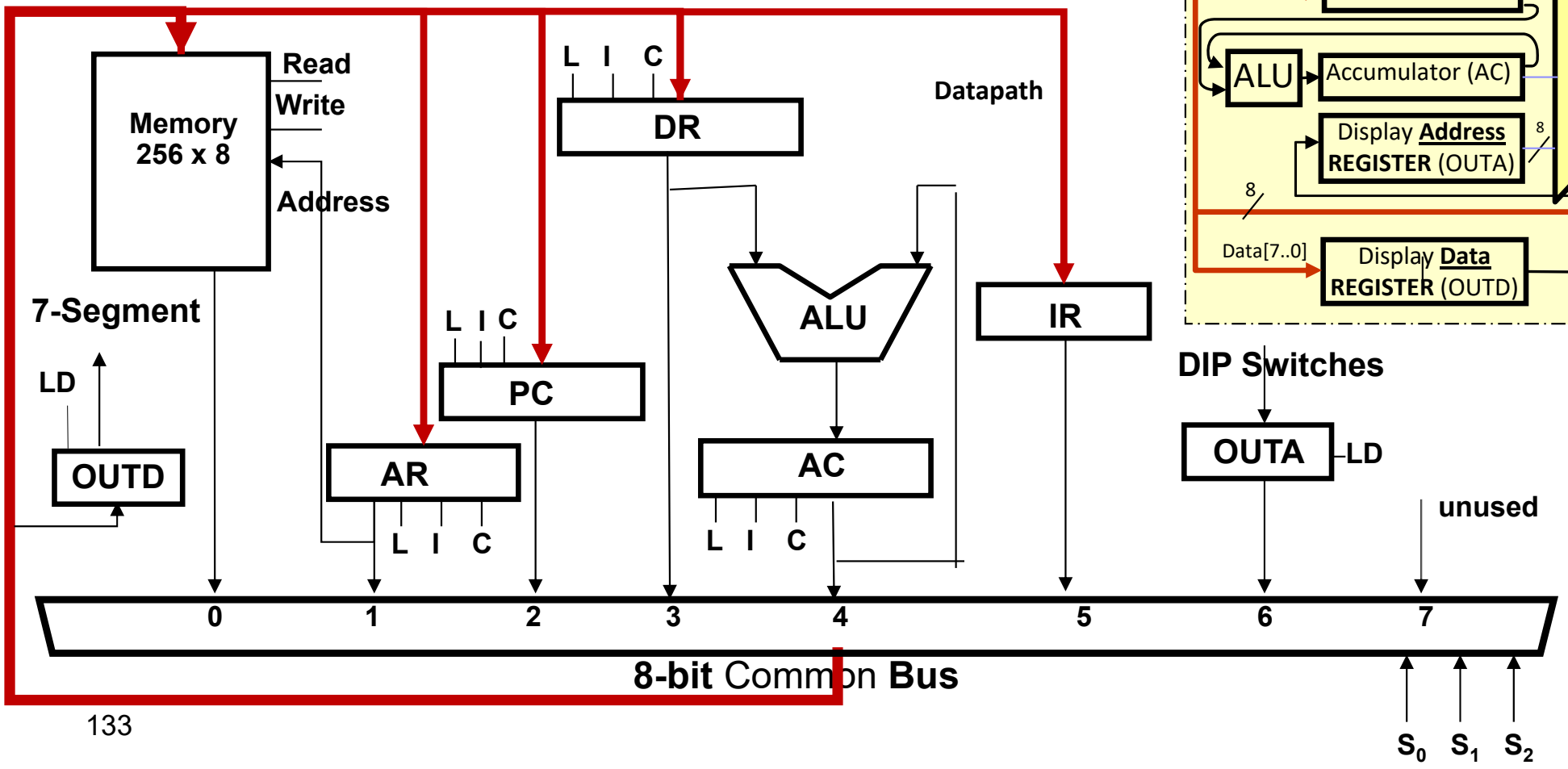
Register reference (I = 0)

Input / Output reference (I =1)

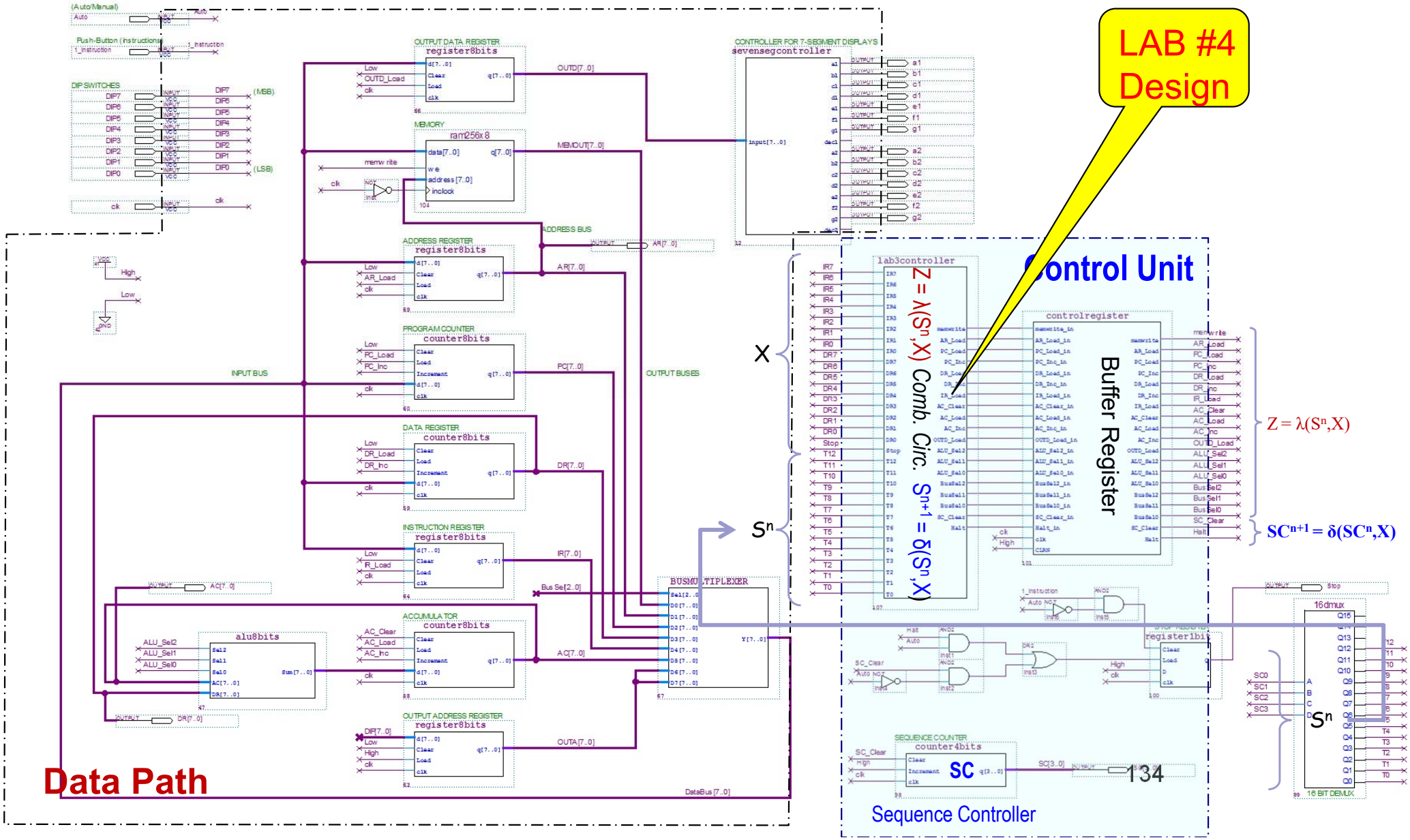


COMMON BUS SYSTEM





Computer Structure



Instructions List & Format

	Symbol	Binary codes = hex I=0 (direct addr) I=1(indirect addr)		Description
Memory Reference	AND	00 000001=01	10 000001=81	AND AC to memory word
	ADD	00 000010=02	10 000010=82	Add a memory word to AC
	SUB	00 000011=03	10 000011=83	Subtract a memory word from AC
	LDA	00 000100=04	10 000100=84	Load AC from a mm01y location
	STA	00 001000=08	10 001000=88	Store AC to a memory location
	BUN	00 010000=10	10 010000=90	Branch unconditionally
	ISZ	00 100000=20	10 100000=AO	Increment counter of memory location and skip the following instruction if the incremented number is 0
Register Reference	CLA	01 000001=41		Clear AC
	CMA	01 000010=42		Complement AC
	ASL	01 000100=44		Arithmetic left shift AC
	ASR	01 001000=48		Arithmetic right shift AC
	INC	01 010000=50		Increment AC
	HLT	01 100000=60		Halt. A Stop bit is set to 1, which prevents PC from being incremented.

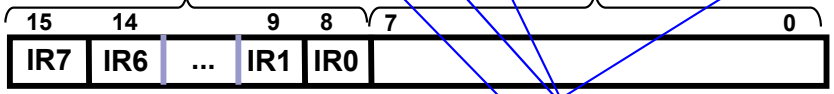
Memory Reference Instruction
Register Ref. Instrc.

OpCode
Operand Address
OpCode

RTL ASM#1

		IR 7	IR 6	IR 5	IR 4	IR 3	IR 2	IR 1	IR 0
		I	Mem/ Reg	O	p	C	o	d	e
Memory Reference	Direct Addr X_0	0	0	x	x	x	x	x	x
	Indirect Addr. X_2	1	0	x	x	x	x	x	x
Reg. Ref.	Instr. X_0	x	1	x	x	x	x	x	x

Memory-reference instruction format:



Direct Addressing (IR7 =0) **Memory reference (IR6 =0)**
Indirect Addressing (IR7 =1) **Register reference (IR6 =1)**

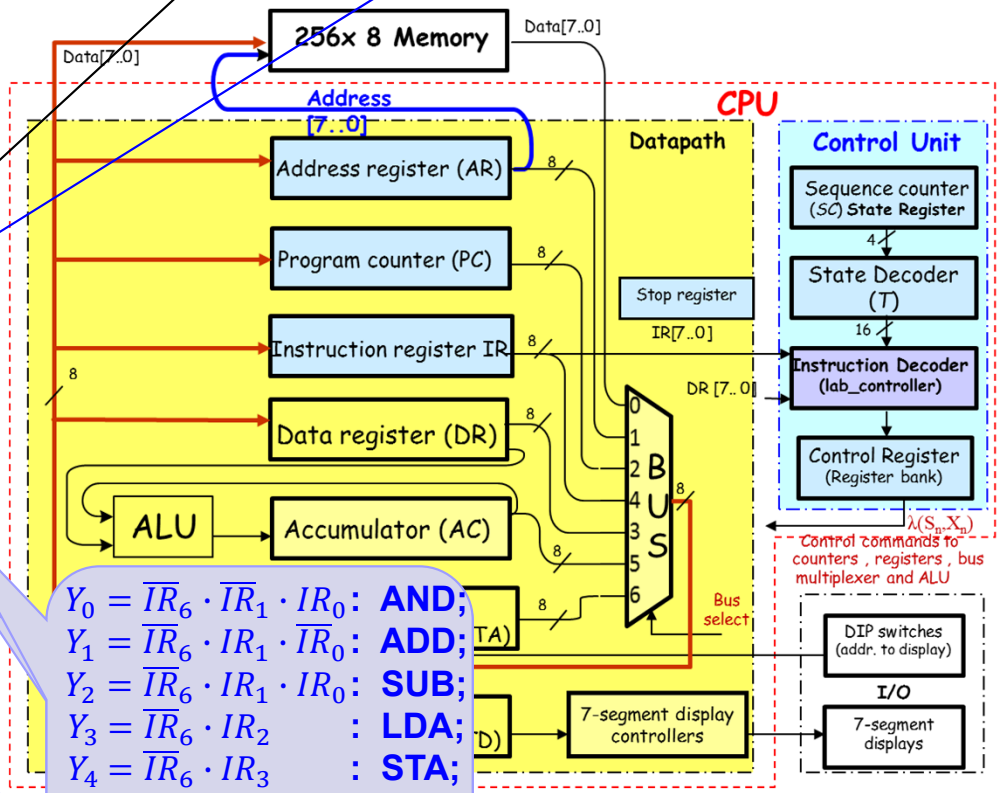
$X_0 = \overline{IR}_7 \cdot \overline{IR}_6$ indicates a direct memory-reference instruction;

$X_1 = \overline{IR}_7 \cdot IR_6$ indicates a register-reference instruction

$X_2 = IR_7 \cdot \overline{IR}_6$ indicates an indirect memory-reference instruction.

There are no input/output instructions:

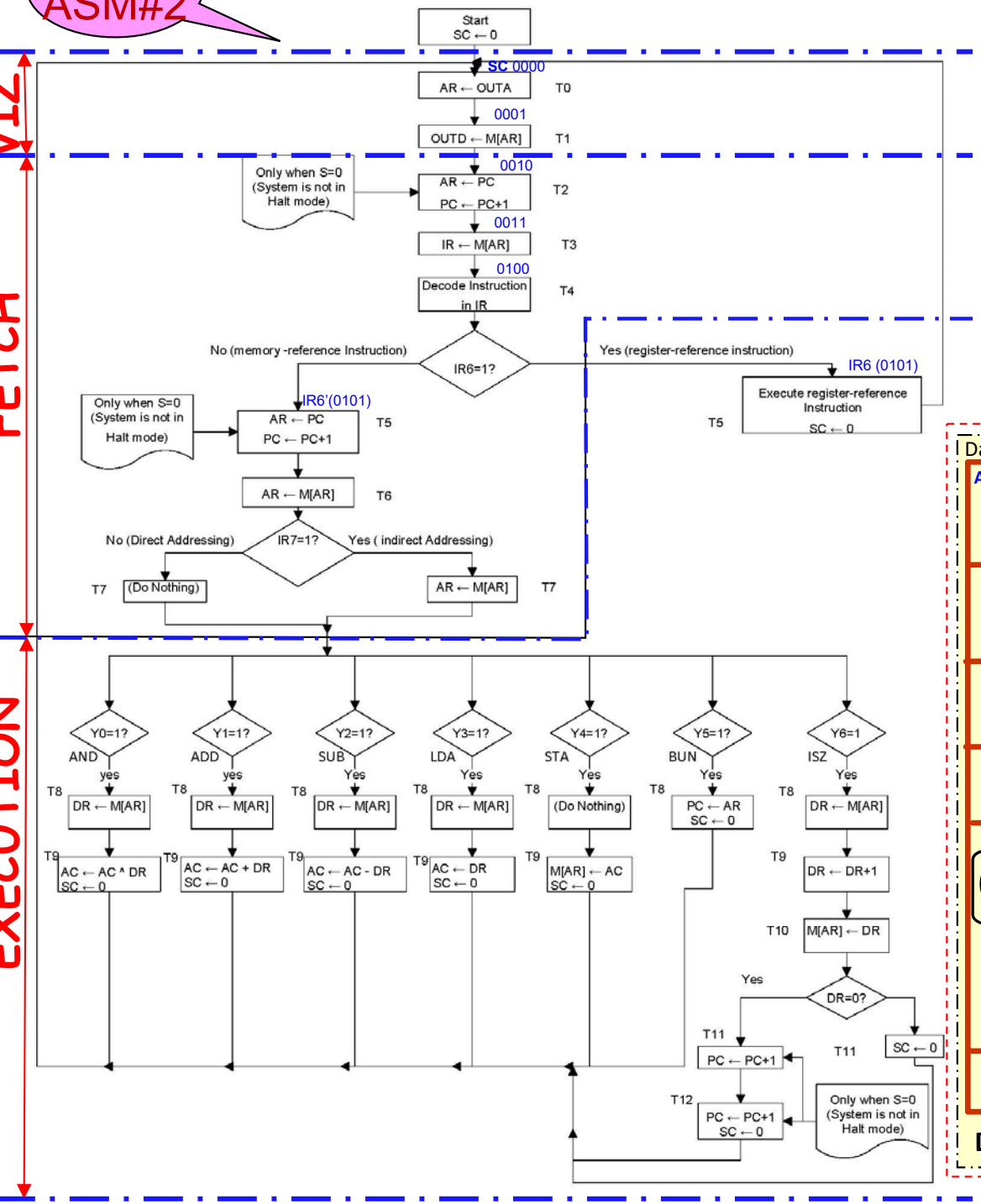
1. The program and data/operands are loaded into the memory when the bit-stream is downloaded to the FPGA.
2. The memory content from an address provided by the DIP switches is visualized on the 7-segment display, while the program is running.



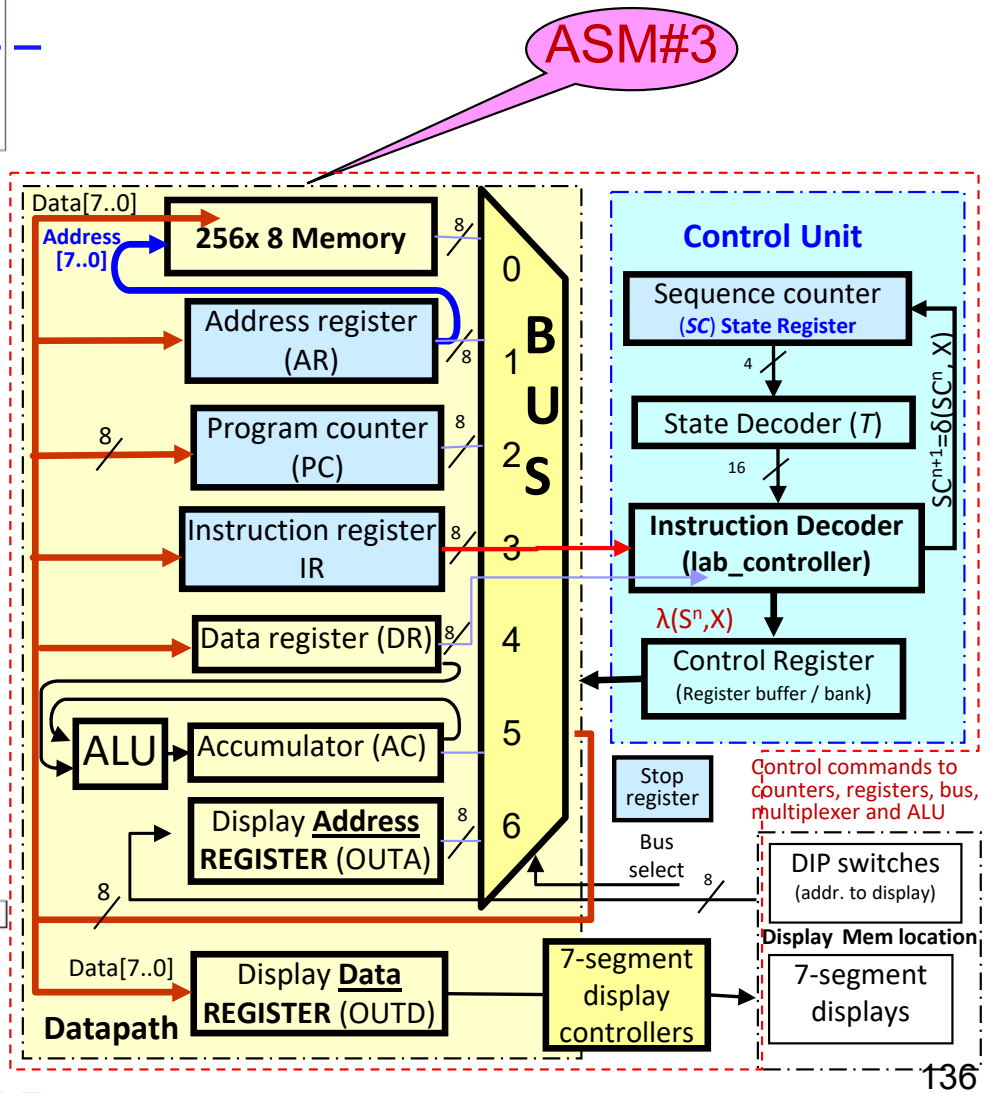
$Y_0 = \overline{IR}_6 \cdot \overline{IR}_1 \cdot \overline{IR}_0$: **AND;**
$Y_1 = \overline{IR}_6 \cdot IR_1 \cdot \overline{IR}_0$: **ADD;**
$Y_2 = \overline{IR}_6 \cdot IR_1 \cdot IR_0$: **SUB;**
$Y_3 = \overline{IR}_6 \cdot IR_2$: **LDA;**
$Y_4 = \overline{IR}_6 \cdot IR_3$: **STA;**
$Y_5 = \overline{IR}_6 \cdot IR_4$: **BUN**
$Y_6 = \overline{IR}_6 \cdot IR_5$: **ISZ**

Instruction Cycle RTL

RTL ASM#2



ASM#3



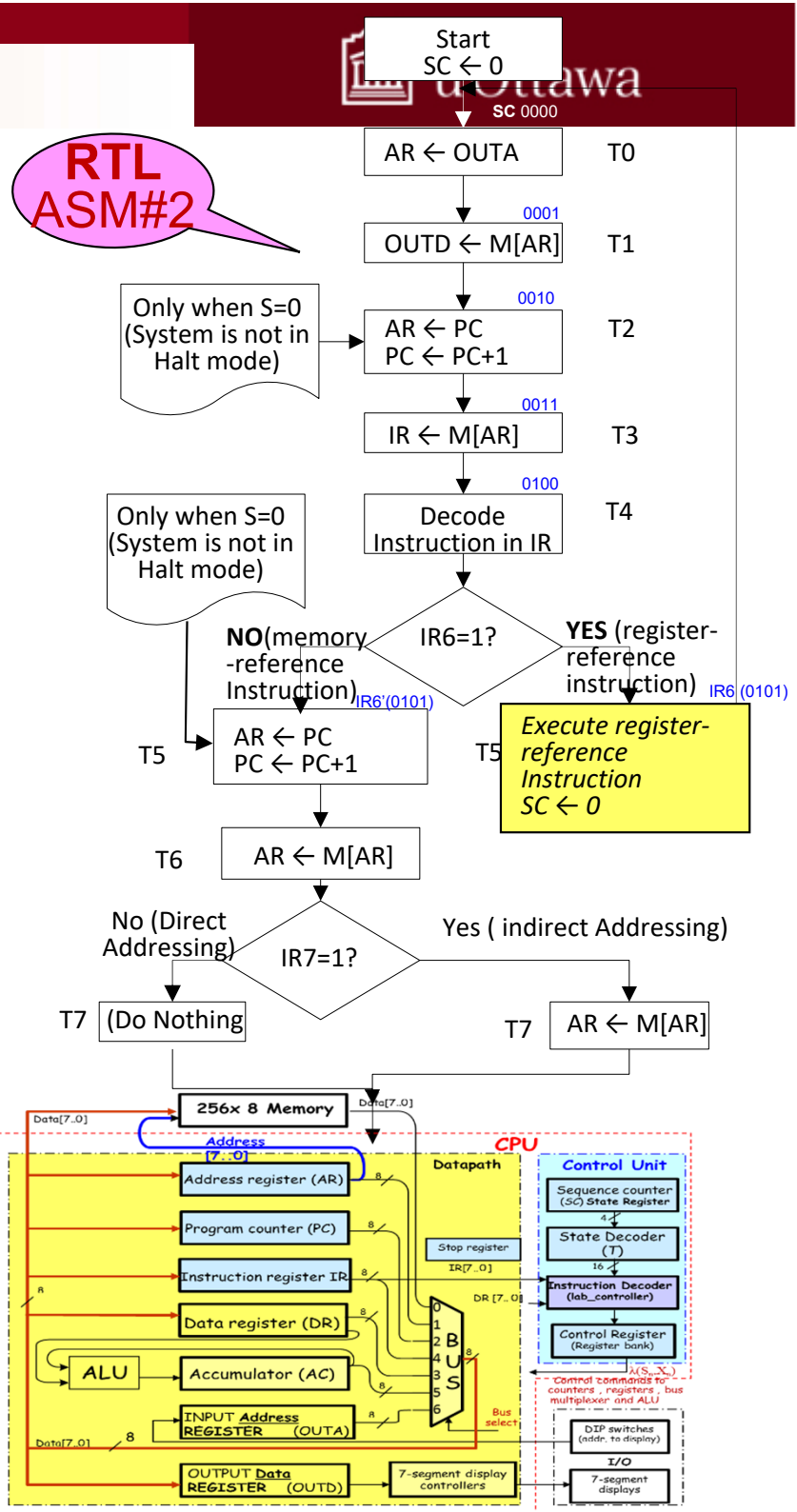
Instruction Fetch + Viz Cycle RTL (ASM #2) - common to all types of instruction

VIZ

FETCH

EXECUTION

State	Description	Notation RTL
T_0	Load the address register AR with the contents of OUTA	T_0: AR ← OUTA
T_1	Read memory location pointed to by AR to register OUTD	T_1: OUTD ← M [AR]
T_2	▪ Load AR register with the ADDRESS of the opcode of the current instruction (PC) ▪ Increment PC (if the program is running, i.e., Stop FF S = 0) to point to the address of the next byte to be read, which can be: ○ the next instruction, if the current instruction is a register-reference instruction ○ the 2nd byte of the current instruction, if it is a memory-reference instruction.	T_2: AR ← PC T_2S': PC ← PC + 1
T_3	Read the instruction's opcode from memory to IR	T_3 : IR ←M [AR]
T_4	This state is a delay that allows the opcode to be decoded in Control Unit.	(nothing)
T_5	*If X_1 ➔ it is a register - reference instruction which will be executed now*	*T_5X_1: exec. Table 3 instruction T_5X_1 :SC ← 0*
	If X_0 or X_2(memory - reference instruction), ▪copy PC to AR, i.e, the address of the instruction's 2nd byte goes from PC to AR => now AR contains the ADDRESS of ○ the operand address if direct addressing, or ○ the address of the operand address if indirect addressing ▪Increment PC if program is running (Stop FF S=0). Note that $(X_0 + X_2) = IR_6$.	$T_5IR'_6$: AR ← PC $T_sIR'_6S'$: PC ← PC + 1
T_6	Read from memory location pointed to by AR to AR; the read byte is - the operand address, if direct addressing, or -the address of the operand address, if indirect addressing	$T_6IR'_6$: AR ← M [AR]
T_7	If indirect addressing, read the operand address from memory location pointed to by AR	T_7X_2 : $AR \leftarrow M[AR]$
	If direct addressing, don't do anything, as the operand address is already in AR since T_6	T_7X_0: (nothing)
T_8 & after	Execute the memory-reference instruction (Table 4).	(see Table 4)



Instructions Execution RTL (ASM #2)

Type of Instruction	Symbol	Binary codes = hex I=0 (direct addressing) I=1(indirect addressing)		Description	RTL Notation
Memory Reference (Table 4)	AND	00 000001=01	10 000001=81	AND AC to memory word	$T_5Y_0: DR \leftarrow M[AR]$ $T_6Y_0: AC \leftarrow AC \wedge DR, SC \leftarrow 0$
	ADD	00 000010=02	10 000010=82	Add a memory word to AC	$T_5Y_1: DR \leftarrow M[AR]$ $T_6Y_1: AC \leftarrow AC + DR, SC \leftarrow 0$
	SUB	00 000011=03	10 000011=83	Subtract a memory word from AC	$T_5Y_2: DR \leftarrow M[AR]$ $T_6Y_2: AC \leftarrow AC - DR, SC \leftarrow 0$
	LDA	00 000100=04	10 000100=84	Load AC from a mm01y location	$T_5Y_3: DR \leftarrow M[AR]$ $T_6Y_3: AC \leftarrow DR, SC \leftarrow 0$
	STA	00 001000=08	10 001000=88	Store AC to a memory location	T_5: cycle not allocated to allow address to stabilize $T_6Y_4: M[AR] \leftarrow AC, SC \leftarrow 0$
	BUN	00 010000=10	10 010000=90	Branch unconditionally	$T_5Y_5: PC \leftarrow AR, SC \leftarrow 0$
	ISZ	00 100000=20	10 100000=*AO*	Increment counter of memory location and skip the following instruction if the incremented number is 0	T_8Y_6: $DR \leftarrow M[AR]$ T_9Y_6: $DR \leftarrow DR + 1$ $T_{10}Y_6$: $M[AR] \leftarrow DR$ $T_{11}Y_6$: if $(DR = 0)$ then $(\bar{S}: PC \leftarrow PC+1)$ $T_{12}Y_6$: if $(DR = 0)$ then $(\bar{S}: PC \leftarrow PC+1), SC \leftarrow 0$
Register Reference (Table 3)	CLA	01 000001=41		Clear AC	$T_5X_1IR_0: AC \leftarrow 0$
	CMA	01 000010=42		Complement AC	$T_5X_1IR_1: AC \leftarrow \overline{AC}$
	ASL	01 000100=44		Arithmetic left shift AC	$T_5X_1IR_2: AC \leftarrow \text{ashl } AC$
	ASR	01 001000=48		Arithmetic right shift AC	$T_5X_1IR_3: AC \leftarrow \text{ashr } AC$
	INC	01 010000=50		Increment AC	$T_5X_1IR_4: AC \leftarrow AC + 1$
	HLT	01 100000=60		Halt. A Stop bit is set to 1, which prevents PC from being incremented.	$T_5X_1IR_5: S \leftarrow 1$

Control Signals Generated by CU

Methods for equations' derivation:

1. Directly from
   a) RTL table or ASM#2 chart
   b) ASM#4 chart
2. By mapping ASM#4 to State Table

Memory ($\lambda_M(In, T)$)

1. memwrite

CPU registers ($\lambda_R(In, T)$)

1. AR_Load
2. PC_Load
3. PC_Inc
4. DR_Load
5. DR_Inc
6. IR_Load
7. AC_Clear
8. AC_Load
9. AC_Inc
10. OUTD_Load

CPU ALU ($\lambda_{ALU}(In, T)$)

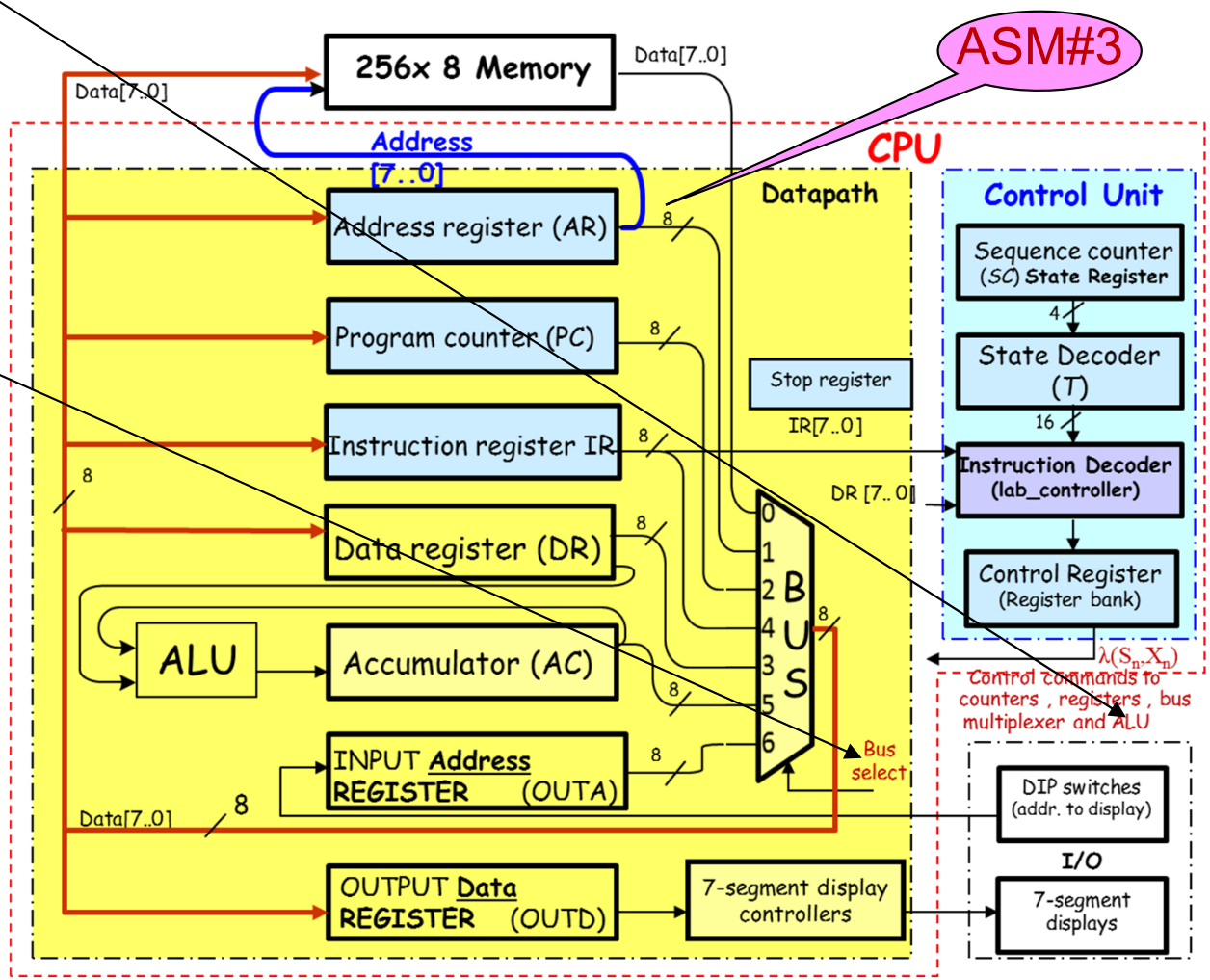
1. ALU_Sel2
2. ALU_Sel1
3. ALU_Sel0

Bus (data mux) ($\lambda_{Bus}(In, T)$)

1. BusSel2
2. BusSel1
3. BusSel0

Control Unit ($\delta(In, T)$)

1. SC_Clear
2. Halt



Description	Notation RTL
Visualization	T_0: AR ← OUTA
	T_1: OUTD ← M [AR]
Fetch	T_2: AR ← PC
	T_2S': PC ← PC + 1
	T_3: IR ← M [AR]
	T_4 - delay that allows the opcode to be decoded
If X_1: **register - reference instruction execute (Table 4)**	
	T_5X_1 :SC ← 0
CLA	$T_5X_1IR_0$: AC ← 0
CMA	$T_5X_1IR_1$: $AC \leftarrow \overline{AC}$
ASL	$T_5X_1IR_2$: AC ← ashl AC
ASR	$T_5X_1IR_3$: AC ← ashr AC
INC	$T_5X_1IR_4$: AC ← AC+ 1
HLT	$T_5X_1IR_5$: S ← 1
If $(X_0 + X_2) = \overline{IR_6}$, **memory - reference instruction fetch operand**	
	$T_5IR'_6$: AR ← PC
	$T_5IR'_6S'$: PC ← PC + 1
	$T_6IR'_6$: AR ← M [AR]
If **indirect** (X_2)	T_7X_2 : AR ← M [AR]
memory - reference instruction execute (Table 3)	
AND	T_8Y_0: DR ← M[AR]
	T_9Y_0: AC ← AC ∧ DR, SC ← 0
ADD	T_8Y_1: DR ← M[AR]
	T_9Y_1: AC ← AC + DR, SC ← 0
SUB	T_8Y_2: DR ← M[AR]
	T_9Y_2: AC ← AC - DR, SC ← 0
LDA	T_8Y_3: DR ← M[AR]
	T_9Y_3: AC ← DR, SC ← 0
STA	T_8: allow the address bus to stabilize
	T_9Y_4: M[AR] ← AC, SC ← 0
BUN	T_8Y_5: PC ← AR, SC ← 0
ISZ (assuming that the next instruction is a memory-reference instruction)	T_8Y_6: DR ← M[AR]
	T_9Y_6: DR ←DR + 1
	$T_{10}Y_6$: M[AR] ←DR
	$T_{11}Y_6$: if (DR = 0) then ($\overline{S}$: PC ← PC+1)
	$T_{12}Y_6$: if (DR = 0) then ($\overline{S}$: PC ← PC+1), SC ← 0

Equations Derivation Method 1a directly from RTL Table

$(\lambda_R(In, T))$ — ASM#5

Scan Table and find all the RTL statements in which

AR is loaded:

AR_Load = $T_0 + T_2 + IR_6' T_5 + T_6IR'_6 + T_7X_2$

PC is loaded:

PC_Load = …

PC is incremented:

PC_Inc = S' T2 + S'IR6' T5 + …

IR is loaded:

IR_Load = T3

OUTD is loaded:

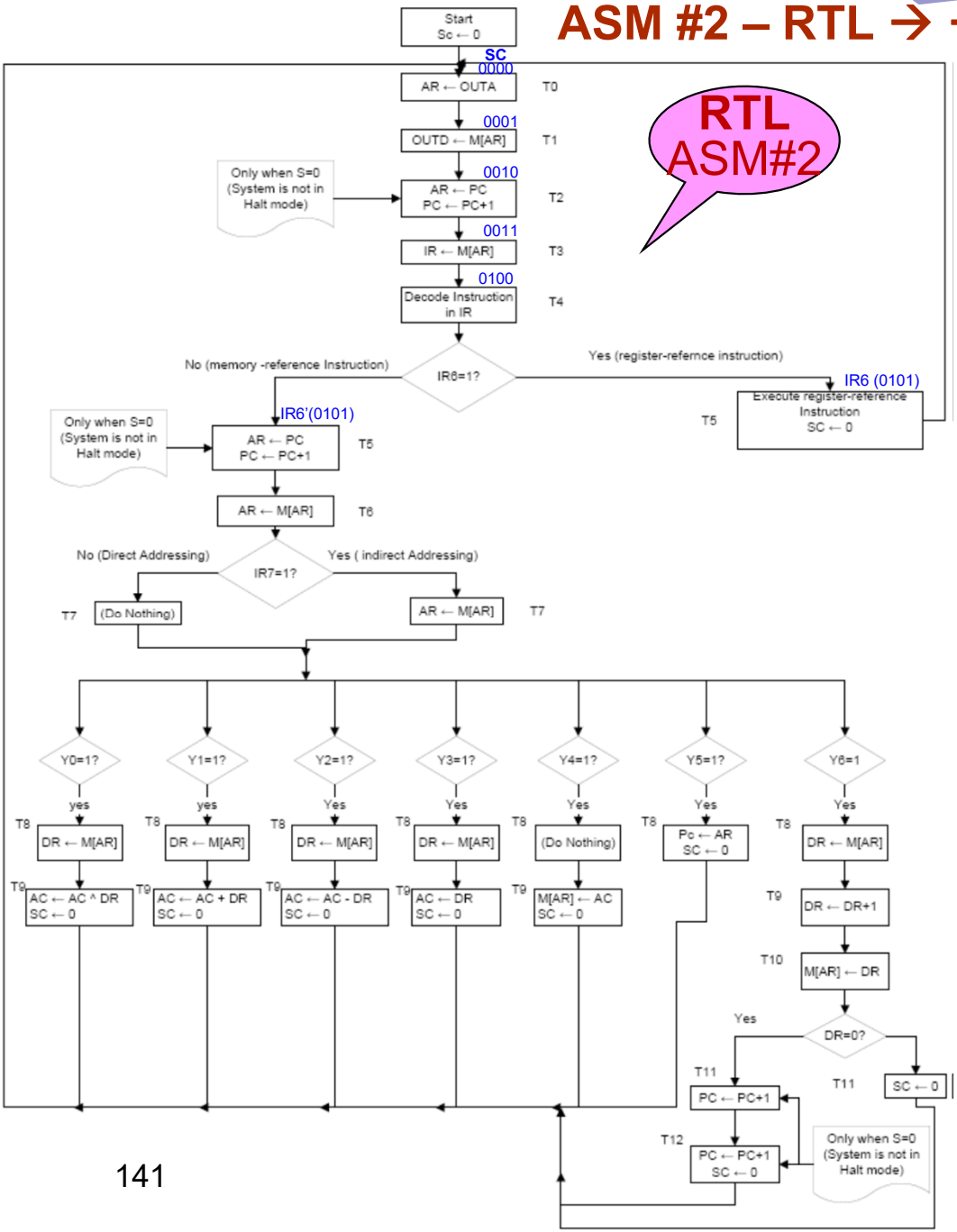
OUTD_Load = T1

….

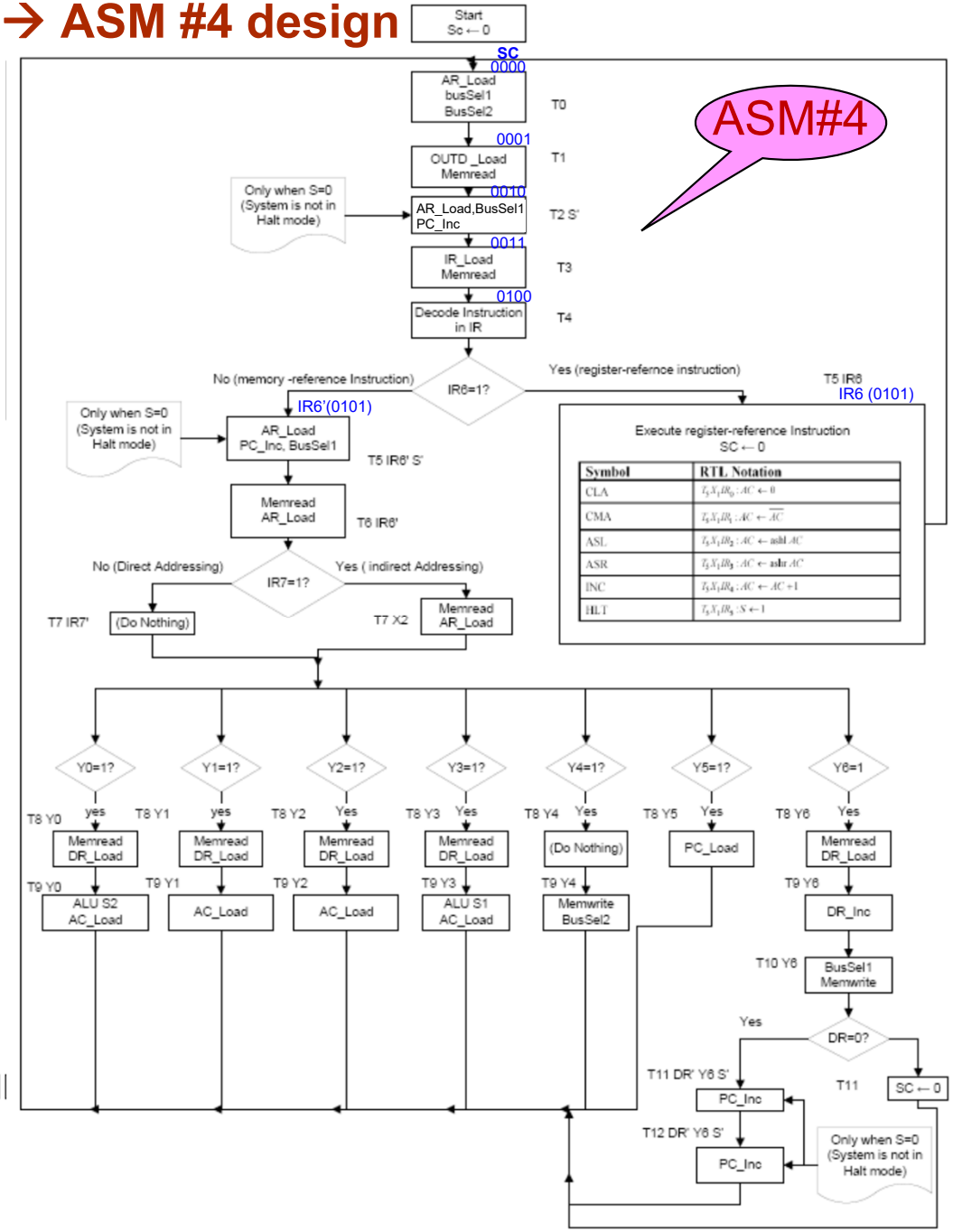
uOttawa

Equations Derivation Method 1b ASM

ASM #2 – RTL → → ASM #4 design



RTL ASM#2



Symbol	RTL Notation
CLA	$T_0X_1IR_0 : AC \leftarrow 0$
CMA	$T_0X_1IR_1 : AC \leftarrow \overline{AC}$
ASL	$T_0X_1IR_2 : AC \leftarrow \text{asl}\, AC$
ASR	$T_0X_1IR_3 : AC \leftarrow \text{asr}\, AC$
INC	$T_0X_1IR_4 : AC \leftarrow AC+1$
HLT	$T_0X_1IR_5 : S \leftarrow 1$

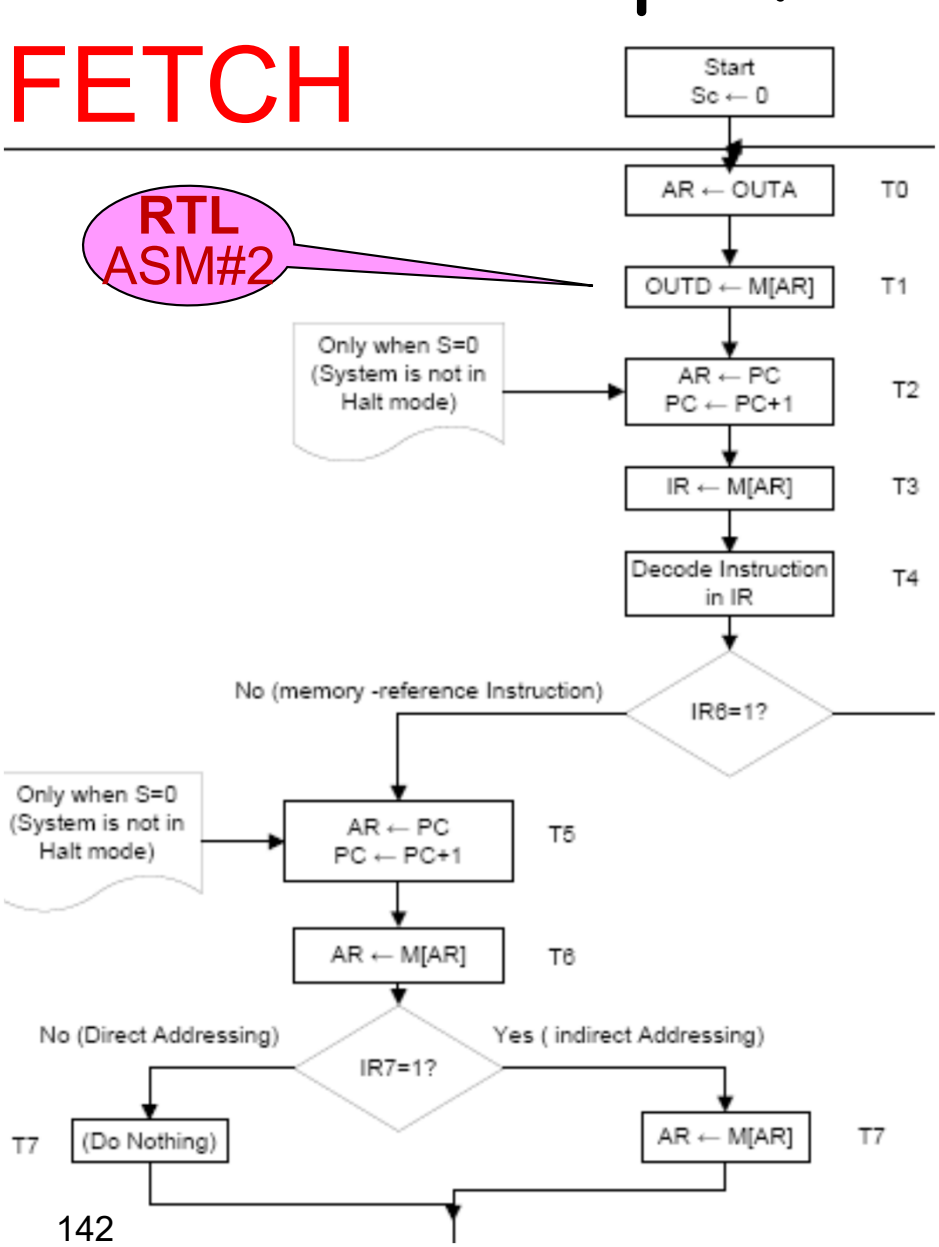
ASM#4

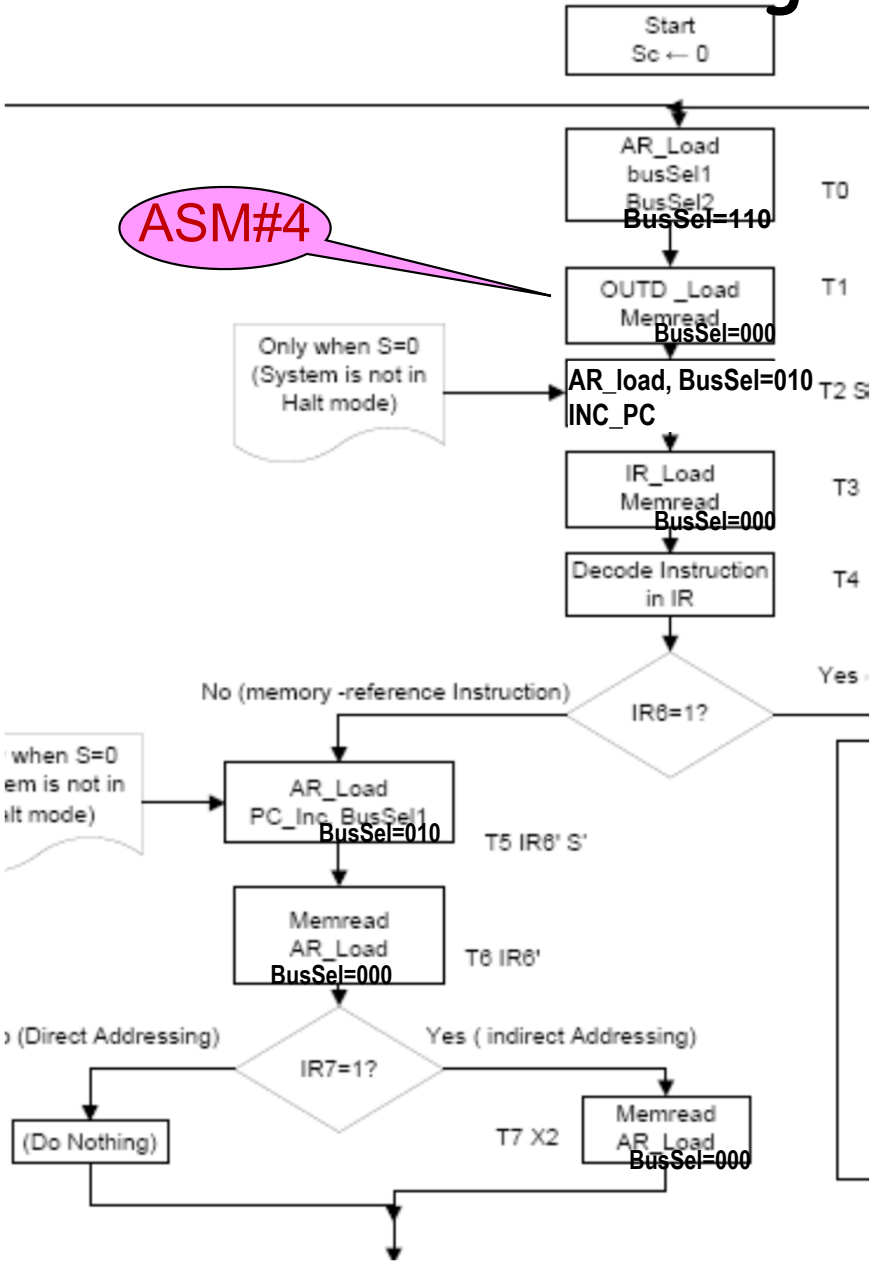
uOttawa

Equations Derivation Method 1b ASM

ASM#2 - RTL specification → → →ASM#4 - design

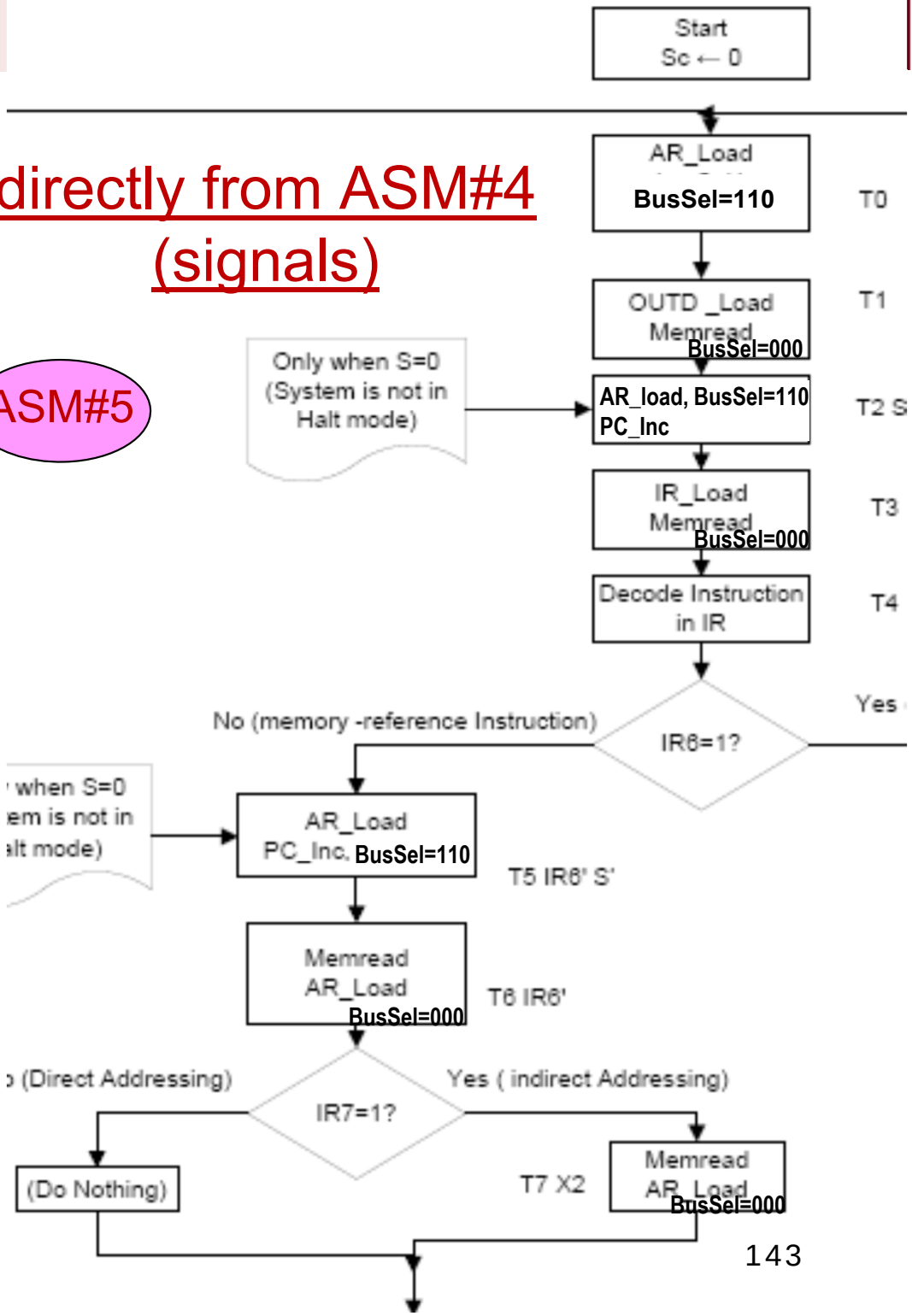
FETCH





Equations Derivation (ASM#5) Method 1b

directly from ASM#4 (signals)



ASM#5

$(\lambda_R(In, T))$

AR_Load = T0 + T2 + IR6′ T5 + …

PC_Load = …

PC_Inc = S′ T2 + IR6′ T5 + …

IR_Load = T3

OUTD_Load = T1

….

$(\lambda_M(In, T))$

Memread = $T_1 + T_3 + T_6\, IR_6 + T_7\, X_2$ + …

…

$(\lambda_{Bus}(In, T))$

BusSel2 = T5 IR6′ S′ + …

BusSel1 = = T5 IR6′ S′ + …

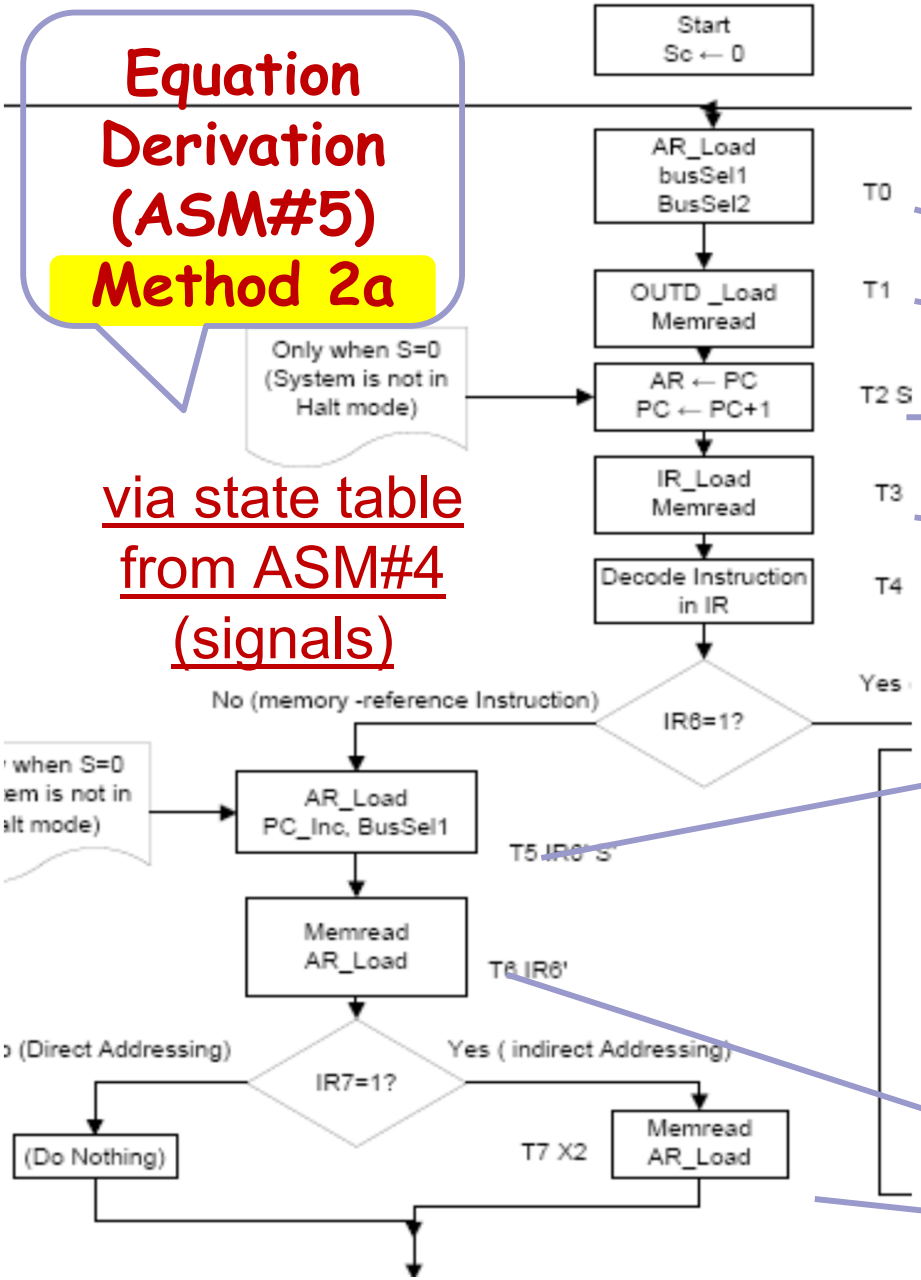
BusSel0 =

RTL ASM→State Table→Equations

ASM#2 → ASM#4 → ASM#5

Equation Derivation (ASM#5) Method 2a

via state table from ASM#4 (signals)



RTL ASM#2

ASM#4

Signals λ

Present State	CU Inputs		Micro-operation	CU Outputs	
Timing signal	Conditions		RTL	Bus Sel	Register Control Signals
T0			AR ← OUTA	110	AR_Load
T1			OUTD ← M[AR]	000	OUTD_Load Memread
T2			AR ← PC	010	AR_Load
	S		PC ← PC+1		PC_Inc
T3			IR ← M[AR]	000	IR_Load Memread
T4			Delay for IR decoding		
T5	IR6'		AR ← PC PC ← PC+1	010	AR_Load PC_Inc
	IR6	IR0	AC←0 (***CLA***)		AC_Clear
		IR1	AC←AC'(***CMA***)		
		IR2			
		IR4			AC_Inc
		IR5			
T6					
T7	IR7				

Memory Control Signals

Equations Derivation (ASM#5) Method 2a

Present State	CU Inputs		Micro-operation	CU Outputs	
Timing signal	Conditions		RTL	Bus Sel	Register Control Signals
T0			AR ← OUTA	110	AR_Load
T1			OUTD ← M[AR]	000	OUTD_Load Memread
T2			AR ← PC	010	AR_Load
	S		PC ← PC+1		PC_Inc
T3			IR ← M[AR]	000	IR_Load Memread
T4			Delay for IR decoding		
T5	IR6'		AR ← PC PC ← PC+1	010	AR_Load PC_Inc
	IR6	IR0	AC←0 (***CLA***)		AC_Clear
		IR1	AC←AC'(***CMA***)		
		IR2			
		IR4			AC_Inc
		IR5			
T6					
T7	IR7				

$(\lambda_M(In, T))$

Memwrite = …

Memread = $T_1 + T_3 +$

ASM#5

RTL ASM→**State Table→Equations**

ASM#2 → **ASM#4 → ASM#5**

Registers Control Signals

Equations Derivation (ASM#5) Method 2a

$(\lambda_R(In, T))$

Present State	CU Inputs		Micro-operation	CU Outputs	
Timing signal	Conditions		RTL	Bus Sel	Register Control Signals
T0			AR ← OUTA	110	AR_Load
T1			OUTD ← M[AR]	000	OUTD_Load Memread
T2			AR ← PC	010	AR_Load
	S		PC ← PC+1		PC_Inc
T3			IR ← M[AR]	000	IR_Load Memread
T4			Delay for IR decoding		
T5	IR6'		AR ← PC PC ← PC+1	010	AR_Load PC_Inc
	IR6	IR0	AC←0 (***CLA***)		AC_Clear
		IR1	AC←AC'(***CMA***)		
		IR2			
		IR4			AC_Inc
		IR5			
T6					
T7	IR7				

RTL ASM→**State Table→Equations**
ASM#2 → **ASM#4** → **ASM#5**

$AR\_Load = T_0 + T_2 + \overline{IR_6}\, T_5 + \ldots.$

$PC\_Load = \ldots..$

$PC\_Inc = \overline{S}\, T_2 + \overline{IR_6}\, T_5 + \ldots$

$DR\_Load = \ldots.$

$DR\_Inc = \ldots.$

$IR\_Load = \ldots.$

$AC\_Clear = T_5\, X_1 IR_0$

$AC\_Load = \ldots.$

$AC\_Inc = \ldots.$

$OUTD\_Load = T_1$

Equations Derivation (ASM#5) Method 2a

RTL ASM→State Table→Equations
ASM#2 → ASM#4 → ASM#5

Present State	CU Inputs		Micro-operation	CU Outputs	
Timing signal	Conditions		RTL	Bus Sel	Register Control Signals
T0			AR ← OUTA	110	AR_Load
T1			OUTD ← M[AR]	000	OUTD_Load Memread
T2			AR ← PC	010	AR_Load
	S		PC ← PC+1		PC_Inc
T3			IR ← M[AR]	000	IR_Load Memread
T4			Delay for IR decoding		
T5	IR6'		AR ← PC PC ← PC+1	010	AR_Load PC_Inc
	IR6	IR0	AC←0 (***CLA***)		AC_Clear
		IR1	AC←AC'(***CMA***)		
		IR2			
		IR4			AC_Inc
		IR5			
T6					
T7	IR7				

State Table → COMMON BUS SYSTEM CONTROL Signals ($\lambda_{Bus}(In, T)$)

Circuit which writes on the bus	Control Conditions (In, T)	Bus select = λ_{Bus}(In, T)		
		BusSel 2	BusSel 1	BusSel 0
Memory	T_1 T_3 $T_6\, IR_6'$ $T_7 X_2$ $T_8 (Y_0 + Y_1 + Y_2 + Y_3 + Y_6)$	0	0	0
AR	$T_8\, Y_5$	0	0	1
PC	T_2 $T_5\, IR_6'$	0	1	0
DR	$T_{10}\, Y_6$	0	1	1
IR	(never happens here)	1	0	0
AC	$T_9\, Y_4$	1	0	1
OUTA	T_0	1	1	0
not used	(indifferent)	1	1	1

$BusSel_0 = \ldots$

$BusSel_1 =$

$BusSel_2 = T_9\, Y_4 + T_0$

ALU selection control signals

Equations Derivation (ASM#5) Method 2a

ALU operations table $(\lambda_{ALU}(In, T))$

ALU_Sel2 S2	ALU_Sel1 S1	ALU_Sel0 S0	Operation	Description
0	0	0	AC + DR	Addition
0	0	1	AC + DR' + 1	Subtraction: AC - DR
0	1	0	ashl AC	AC arithmetic left shift
0	1	1	ashr AC	AC arithmetic right shift
1	0	0	AC ∧ DR	logic AND
1	0	1	AC ∨ DR	logic OR
1	1	0	DR	DR transfer
1	1	1	AC'	Complement AC

RRI Table 3

Symbol	RTL Notation
CLA	$T_5X_1IR_0 : AC \leftarrow 0$
CMA	$T_5X_1IR_1 : AC \leftarrow \overline{AC}$
ASL	$T_5X_1IR_2 : AC \leftarrow \text{ashl } AC$
ASR	$T_5X_1IR_3 : AC \leftarrow \text{ashr } AC$
INC	$T_5X_1IR_4 : AC \leftarrow AC + 1$
HLT	$T_5X_1IR_5 : S \leftarrow 1$

MRI Table 4

Symbol	RTL Notation
AND	$T_8Y_0 : DR \leftarrow M[AR]$ $T_9Y_0 : AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	$T_8Y_1 : DR \leftarrow M[AR]$ $T_9Y_1 : AC \leftarrow AC + DR, SC \leftarrow 0$
SUB	$T_8Y_2 : DR \leftarrow M[AR]$ $T_9Y_2 : AC \leftarrow AC - DR, SC \leftarrow 0$
LDA	$T_8Y_3 : DR \leftarrow M[AR]$ $T_9Y_3 : AC \leftarrow DR, SC \leftarrow 0$

ALU_Sel0 = S0 = $T_9\ Y_2 + T_5\ X_1\ IR_3 + \ldots$

ALU_Sel1 = S1 = …

ALU_Sel2 = S2 = …

RTL ASM→**State Table→Equations**

ASM#2 → **ASM#4 → ASM#5**

Instruction Phase	Timing	State	Notation RTL micro-op (Table 2)	BusSel2	BusSel1	BusSel0	FROM / TO		ALU_Sel2	ALU_Sel1	ALU_Sel0	memwrite	AR_Load	PC_Load	PC_Inc	DR_Load	DR_Inc	IR_Load	AC_Clear	AC_Load	AC_Inc	OUTD_Load	SC_Clear	Halt
Display	T^0	T^0	AR←OUTA	1	1	X	AR_Load						1											
	T^1	T^1	OUTD←M[AR]	0	0	0	OUTD_Load	Read														1		
Fetch opcode	T^2	T^2	AR←PC	0	1	0	AR_Load						1											
		T^2S'	PC←PC+1				PC_Inc								1									
	T^3	T^3	IR←M[AR]	0	0	0	IR_Load	Read										1						
	T^4	T^4	Nothing																					
Exec Reg-Ref instr (IR^6=1 - Table 3)		$IR'^7\ IR^6=X^1$																						
CLA	T^5	$T^5\ X^1\ IR^0$	AC←0				AC_Clear												1					
CMA		$T^5\ X^1\ IR^1$	AC←AC'	1	0	1	AC_Load		1	1	1									1				
ASL		$T^5\ X^1\ IR^2$	AC←ashl AC																					
ASR		$T^5\ X^1\ IR^3$	AC←ashr AC																					
INC		$T^5\ X^1\ IR^4$	AC←AC+1																					
HLT		$T^5\ X^1\ IR^5$	S←1																					
		$T^5\ X^1$	**SC←0**																					
Fetch 2'nd byte of Mem-Ref instr (IR^6=0)		$IR'^7\ IR'^6=X^0$ $IR^7\ IR'^6=X^2$																						
If Mem-Ref Instr (IR'^6) : addr of instr 2nd byte (of PC) => AR		$T^5\ IR'^6$	AR←PC																					
		$T^5\ IR'^6\ S'$	PC←PC+1																					
Fetch instr 2nd byte=addr field = addr of operand (direct addr), or addr of operand's addr (indir)	T^6	$T^6\ IR'^6$	AR←M[AR]																					
Do nothing (direct addr)	T^7	$T^7\ X^0$	Nothing																					
or fetch operand (indirect addr)		$T^7\ X^2$	AR←M[AR]																					
Exec Mem-Ref instr (IR^6=0 - Table 4)		$IR'^6\ IR'^1\ IR^0=Y^0$ $IR'^6\ IR^1\ IR'^0=Y^1$ $IR'^6\ IR^1\ IR^0=Y^2$ $IR'^6\ IR^2=Y^3$ $IR'^6\ IR^3=Y^4$ $IR'^6\ IR^4=Y^5$ $IR'^6\ IR^5=Y^6$																						
AND	T^8	$T^8\ Y^0$	DR←M[AR]																					
		$T^9\ Y^0$	AC←AC * DR, SC←0																					
ADD		$T^8\ Y^1$	DR←M[AR]																					
		$T^9\ Y^1$	AC←AC + DR, SC←0																					
SUB		$T^8\ Y^2$	DR←M[AR]																					
		$T^9\ Y^2$	AC←AC - DR, SC←0																					
LDA		$T^8\ Y^3$	DR←M[AR]																					
		$T^9\ Y^3$	AC←DR, SC←0																					
STA		$T^9\ Y^4$	M[AR]←AC, SC←0																					
BUN		$T^8\ Y^5$	PC←AR, SC←0																					
ISZ		$T^8\ Y^6$	DR←M[AR]																					
		$T^9\ Y^6$	DR←DR+1																					
		$T^{10}\ Y^6$	M[AR]←DR																					
		$T^{11}\ Y^6$	if (DR=0 & S=0) then (PC ← PC + 1)																					
		$T^{12}\ Y^6$	if (DR=0 & S=0) then (PC ← PC + 1), **SC←0**																					

Equation Derivation Method 2b

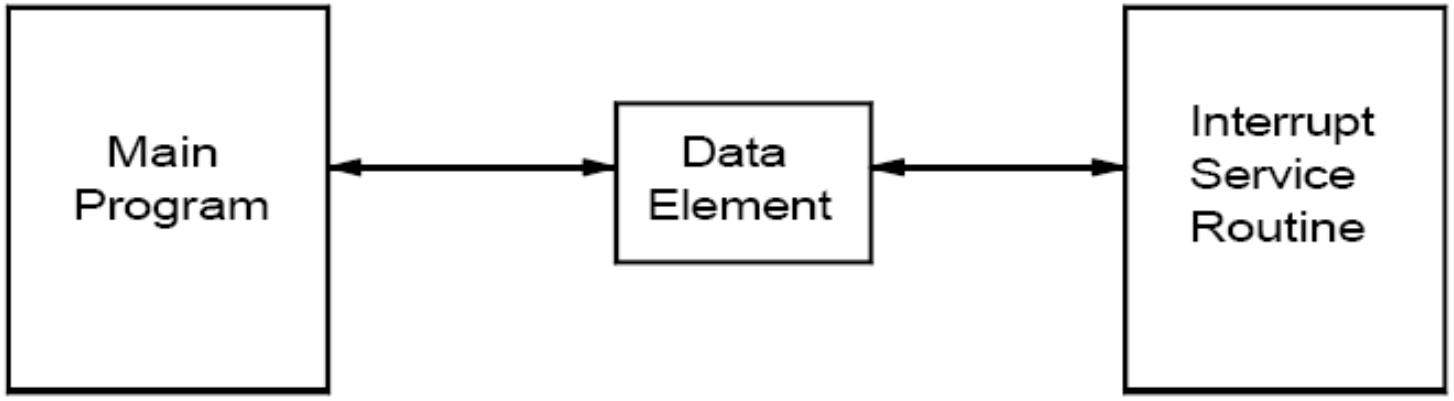
RTL Table → State Table → Equations

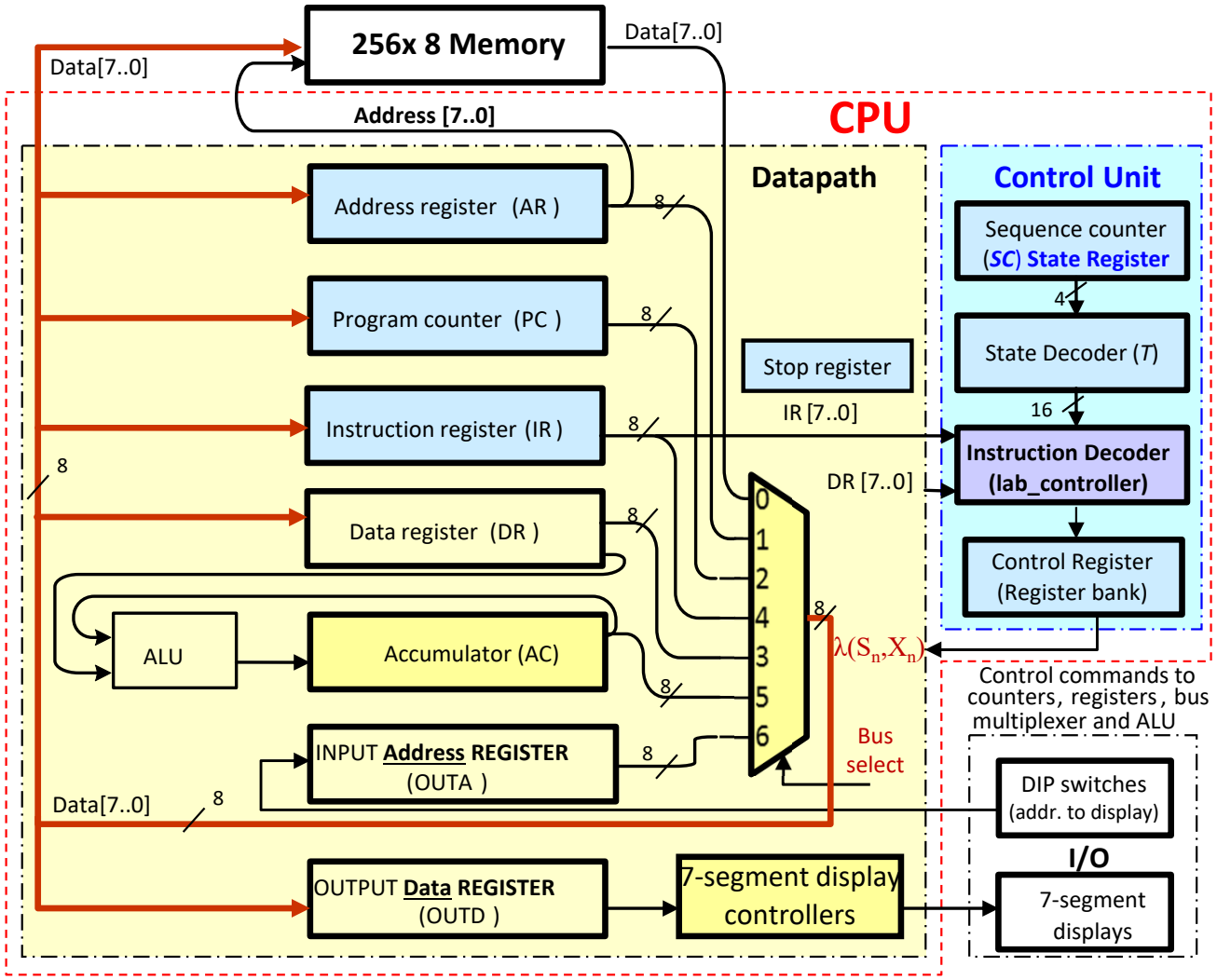
ASM#2 → ASM#4 → ASM#5

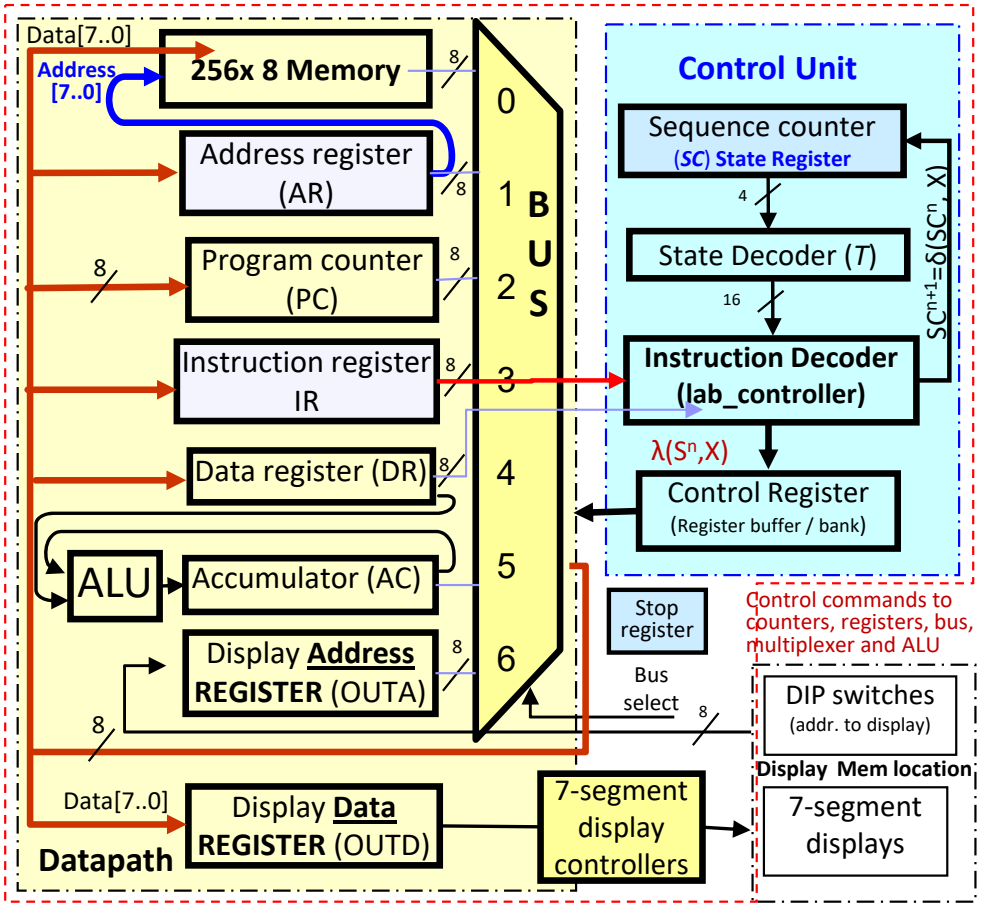
Data Exchange with ISRs

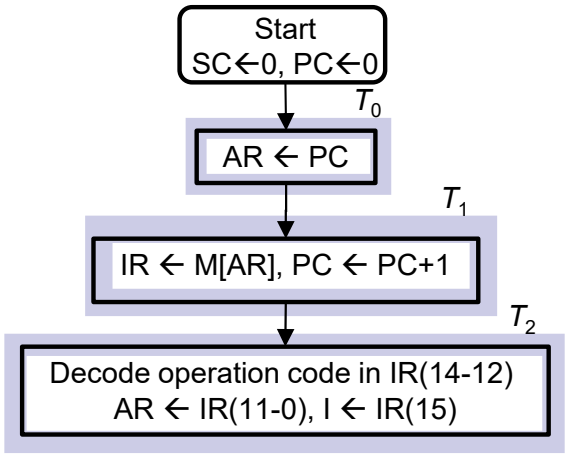
- Use global data
- May need to disable interrupts for critical regions of code using this data

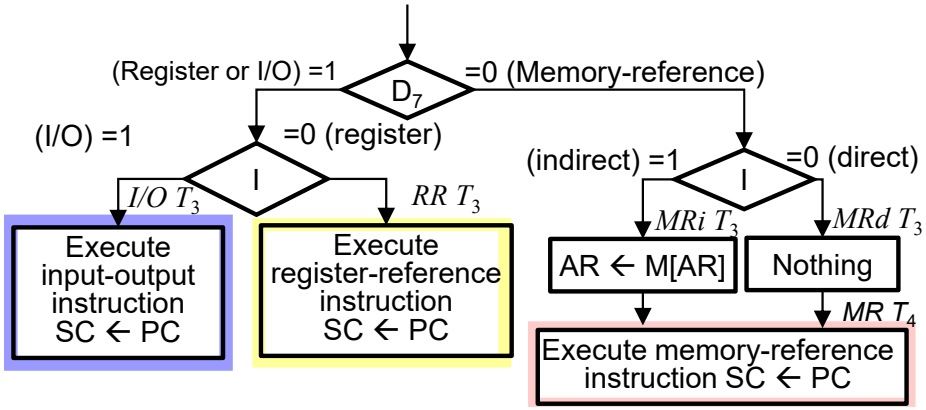
Input-Output and Interrupt

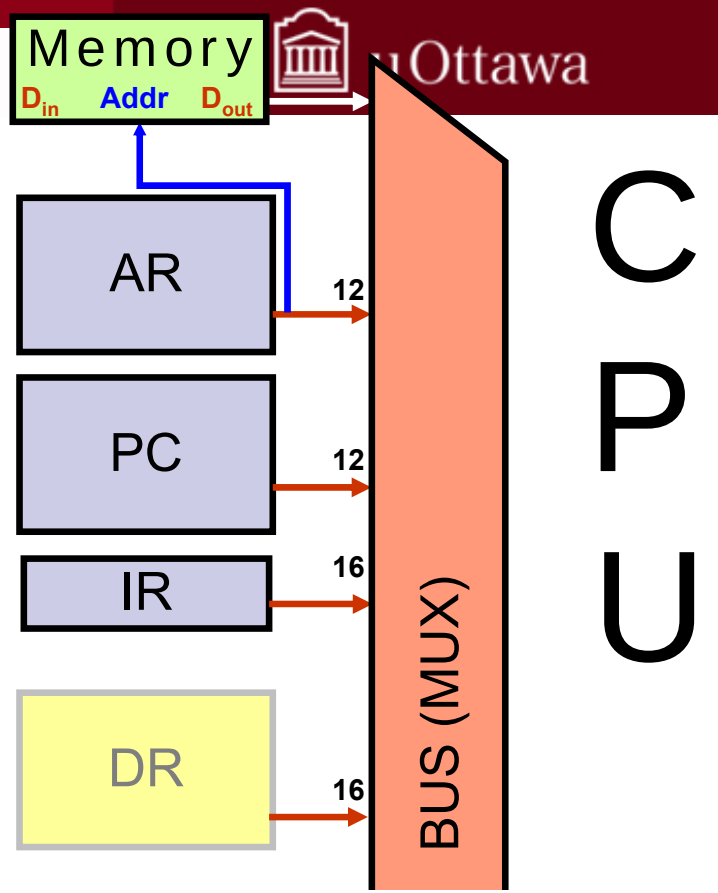


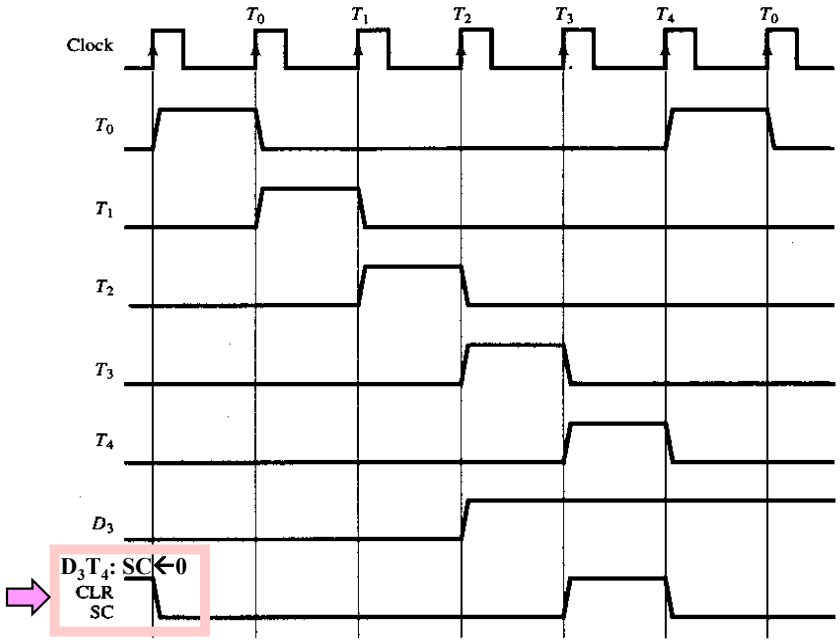

256x 8 Memory
Data[7..0]
Data[7..0]
Address [7..0]
CPU
Datapath
Control Unit
Address register (AR)
Program counter (PC)
Instruction register (IR)
Data register (DR)
Stop register
IR [7..0]
DR [7..0]
Sequence counter (SC) State Register
State Decoder (T)
Instruction Decoder (lab_controller)
Control Register (Register bank)
ALU
Accumulator (AC)
INPUT Address REGISTER (OUTA)
OUTPUT Data REGISTER (OUTD)
Bus select
λ(Sn,Xn)
Control commands to counters, registers, bus multiplexer and ALU
DIP switches (addr. to display)
I/O
7-segment displays
7-segment display controllers
8
4
16


Data[7..0]
Address [7..0]
256x 8 Memory
Address register (AR)
Program counter (PC)
Instruction register IR
Data register (DR)
ALU
Accumulator (AC)
Display Address REGISTER (OUTA)
Data[7..0]
Display Data REGISTER (OUTD)
Datapath
BUS
0
1
2
3
4
5
6
8
Control Unit
Sequence counter (SC) State Register
4
State Decoder (T)
16
Instruction Decoder (lab_controller)
SCn+1=δ(SCn, X)
λ(Sn,X)
Control Register (Register buffer / bank)
Stop register
Control commands to counters, registers, bus, multiplexer and ALU
Bus select
DIP switches (addr. to display)
Display Mem location
7-segment displays
7-segment display controllers


Start
SC←0, PC←0
T_0
AR ← PC
T_1
IR ← M[AR], PC ← PC+1
T_2
Decode operation code in IR(14-12)
AR ← IR(11-0), I ← IR(15)


(Register or I/O) =1
D_7
=0 (Memory-reference)
(I/O) =1
=0 (register)
I
$I/O\ T_3$
$RR\ T_3$
Execute input-output instruction SC ← PC
Execute register-reference instruction SC ← PC
(indirect) =1
I
=0 (direct)
$MRi\ T_3$
$MRd\ T_3$
AR ← M[AR]
Nothing
$MR\ T_4$
Execute memory-reference instruction SC ← PC


Memory
D_{in} Addr D_{out}
AR
PC
IR
DR
12
12
16
16
BUS (MUX)
C
P
U


Clock
T_0
T_1
T_2
T_3
T_4
T_0
T_0
T_1
T_2
T_3
T_4
D_3
D_3T_4: SC←0
CLR SC