

Systems Architecture I

Topics

**Random Access Machines
Implementation of a Simplified Computer**

Notes Courtesy of Jeremy R. Johnson

Systems Architecture I

Topic 1: Random Access Machines

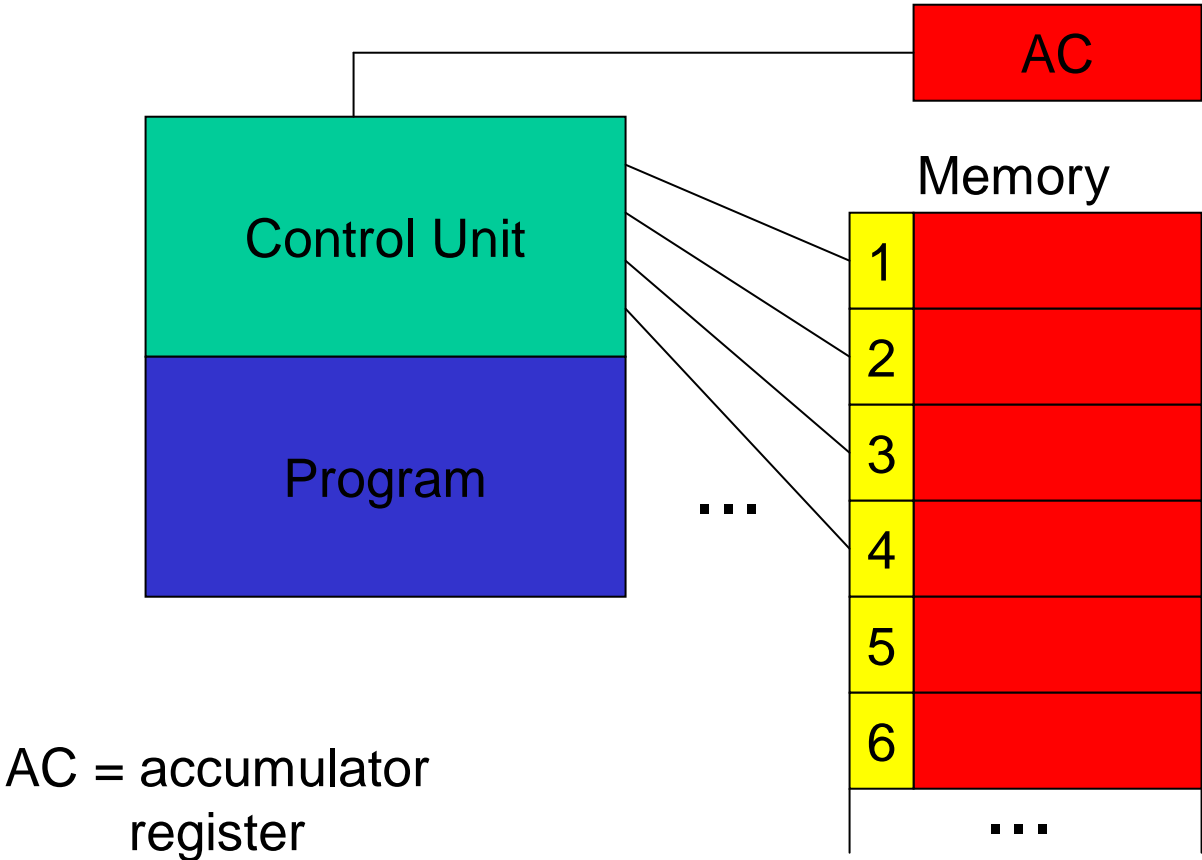
Introduction

- **Objective: To develop a simple model of computation that provides insight into how a program executes on a computer.**
- **A Random Access Machine (RAM) is an abstract model of computation that resembles a simple idealized computer. It is equivalent in computational power to a Turing machine (can perform any computation). Despite its simplicity it provides some intuition as to how a program executes on a computer.**

Definition of a RAM

- **Defined by a set of instructions and a model of execution.**
- **A program for a RAM is a sequence of instructions.**
- **A RAM has an infinite memory. Instructions can read and write to memory. Items from memory are loaded into registers, where arithmetic can be performed.**
- **The state of a computation: program counter (to keep track of instruction to execute), registers, and memory.**

A Random Access Machine



Instruction Set

- **LDA X; Load the AC with the contents of memory address X**
- **LDI X; Load the AC indirectly with the contents of address X**
- **STA X; Store the contents of the AC at memory address X**
- **STI X; Store the contents of the AC indirectly at address X**
- **ADD X; Add the contents of address X to the contents of the AC**
- **SUB X; Subtract the contents of address X from the AC**
- **JMP X; Jump to the instruction labeled X**
- **JMZ X; Jump to the instruction labeled X if the AC contains 0**
- **JMN X; Jump to the instruction labeled X if the contents of the AC ; is negative**
- **HLT ; Halt execution**

Sample Program

STOR

; algorithm to detect duplicates in an array A of size n.

```
for i ← 1 to n do  
  if B(A(i)) ≠ 0  
    then output A(i);  
    exit  
  else B(A(i)) = 1
```

Sample RAM Program

1. LDI 3; get ith entry from A
2. ADD 4; add offset to compute index j
3. STA 5; store index j
4. LDI 5; get jth entry from B
5. JMZ 9; if entry 0, go to 9
6. LDA 3; if entry 1, get index i
7. STA 2; and store it at 2.
8. HLT ; stop execution
9. LDA 1; get constant 1
10. STI 5; and store it in B
11. LDA 3; get index i
12. SUB 4; subtract limit
13. JMZ 8; if i = limit, stop
14. LDA 3; get index i again
15. ADD 1; increment i
16. STA 3; store new value of i
17. JMP 1;

AC

Memory

1	1	constant
2	0	answer
3	6	Index i
4	9	Limit of A
5	0	Index j
6	3	A
7	4	
8	2	
9	2	B
10	0	
11	0	
12	0	
13	0	

Exercises

- **Modify STOR so that when a computation finishes and the input sequence contained a duplicate integer, we know what that integer was.**
- **Modify STOR so that it uses array indexing when accessing the array A instead of pointer arithmetic (i.e. the index into A should be an array index, starting with 1, rather than an address of a location in the array).**
- **Write a RAL program which takes two input integers at addresses 1 and 2 and multiplies them storing the result at address 4.**

Sample Solution

compute $x*y$, $x,y \geq 0$

1. LDA 1; load x
2. JMZ 10; check if x = 0
3. LDA 4; load partial result
4. ADD 2; add y to partial result
5. STA 4; store partial result
6. LDA 1; load x
7. SUB 3; and decrement
8. STA 1; store decremented x
9. JMP 2; next iteration
10. HLT ;



Memory

1	x	value of x
2	y	Value of y
3	1	Constant 1
4	0	result

The program still works with $y < 0$; however, if $x < 0$, it will go into an infinite loop (x will never = 0). To allow $x < 0$, first check to see if x is negative with JMN, and if so we want to increment x rather than decrement it.

Systems Architecture I

Topic 2: Implementation of a Simplified Computer

Introduction

- **Objective: To develop a simple model of a computer and its execution that is capable of executing RAM programs. To introduce the concept of abstraction in computer design.**
- **The model will be given schematically with timing sequences.**
- **RAL instructions will be implemented using microinstructions described in a notation called “Register Transfer Language” (RTL).**
- **The control logic for implementing microinstructions will be described at the gate level.**

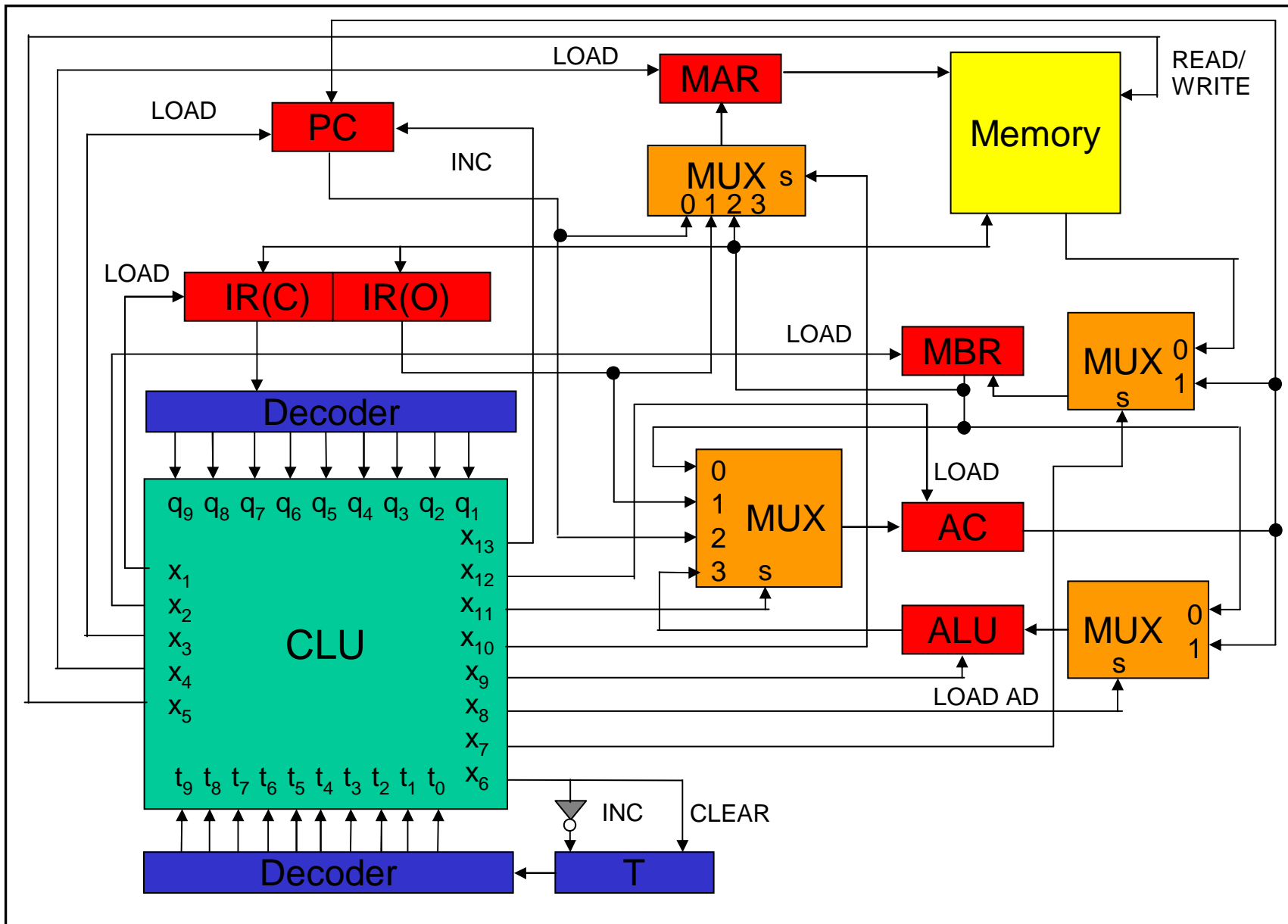
References: Dewdney, The New Turing Omnibus (Chapter 48).

SCRAM

- **A Simple but Complete Random Access Machine. This computer can execute RAL instructions.**
- **8-bit words**
- **16 word memory (4 address bits)**
- **Instructions (4 bit opcode, 4 bit operand)**
- **7 registers**
 - PC (program counter)
 - IR (instruction register - IR(C) = instruction code, IR(O) = operand)
 - MAR (memory address register)
 - MBR (memory buffer register)
 - AC (accumulator)
 - AD (register for addition internal to the ALU - arithmetic logic unit)
- **Driven by the CLU (control logic unit)**
- **A timer T generates pulses that are decoded into separate input lines to the CLU**

Fetch and Execute

- **A cycle of operation consists of two stages**
 - The fetch cycle gets the next executable instruction and loads it into the IR
 - The execute cycle performs the instruction in the IR
- **The fetch and execute cycles are written as a sequence of micro-instructions described in a notation called “Register Transfer Language” (RTL)**



Instruction Opcodes

- **LDA** **0001** **X**; Load contents of memory address X into the AC
- **LDI** **0010** **X**; Indirectly load contents of address X into the AC
- **STA** **0011** **X**; Store contents of AC at memory address X
- **STI** **0100** **X**; Indirectly store contents of AC at address X
- **ADD** **0101** **X**; Add contents of address X to the AC
- **SUB** **0110** **X**; Subtract contents of address X from the AC
- **JMP** **0111** **X**; Jump to the instruction labeled X
- **JMZ** **1000** **X**; Jump to instruction X if the AC contains 0

MicroProgram

- **Fetch cycle**

- t_0 : $MAR \leftarrow PC$
- t_1 : $MBR \leftarrow M$; $PC \leftarrow PC + 1$
- t_2 : $IR \leftarrow MBR$

- **Execute cycle (LDA)**

- q_1t_3 : $MAR \leftarrow IR(O)$
- q_1t_4 : $MBR \leftarrow M$
- q_1t_5 : $AC \leftarrow MBR$

MicroProgram

- **Execute cycle (LDI)**

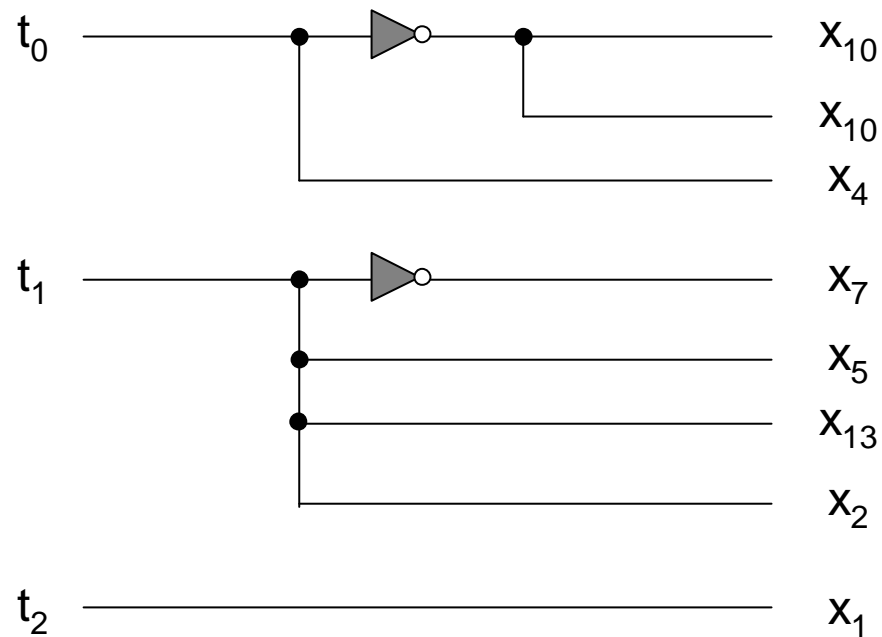
- q_2t_3 : $MAR \leftarrow IR(O)$
- q_2t_4 : $MBR \leftarrow M$
- q_2t_5 : $MAR \leftarrow MBR$
- q_2t_6 : $MBR \leftarrow M$
- q_2t_7 : $AC \leftarrow MBR$

- **Execute cycle (ADD)**

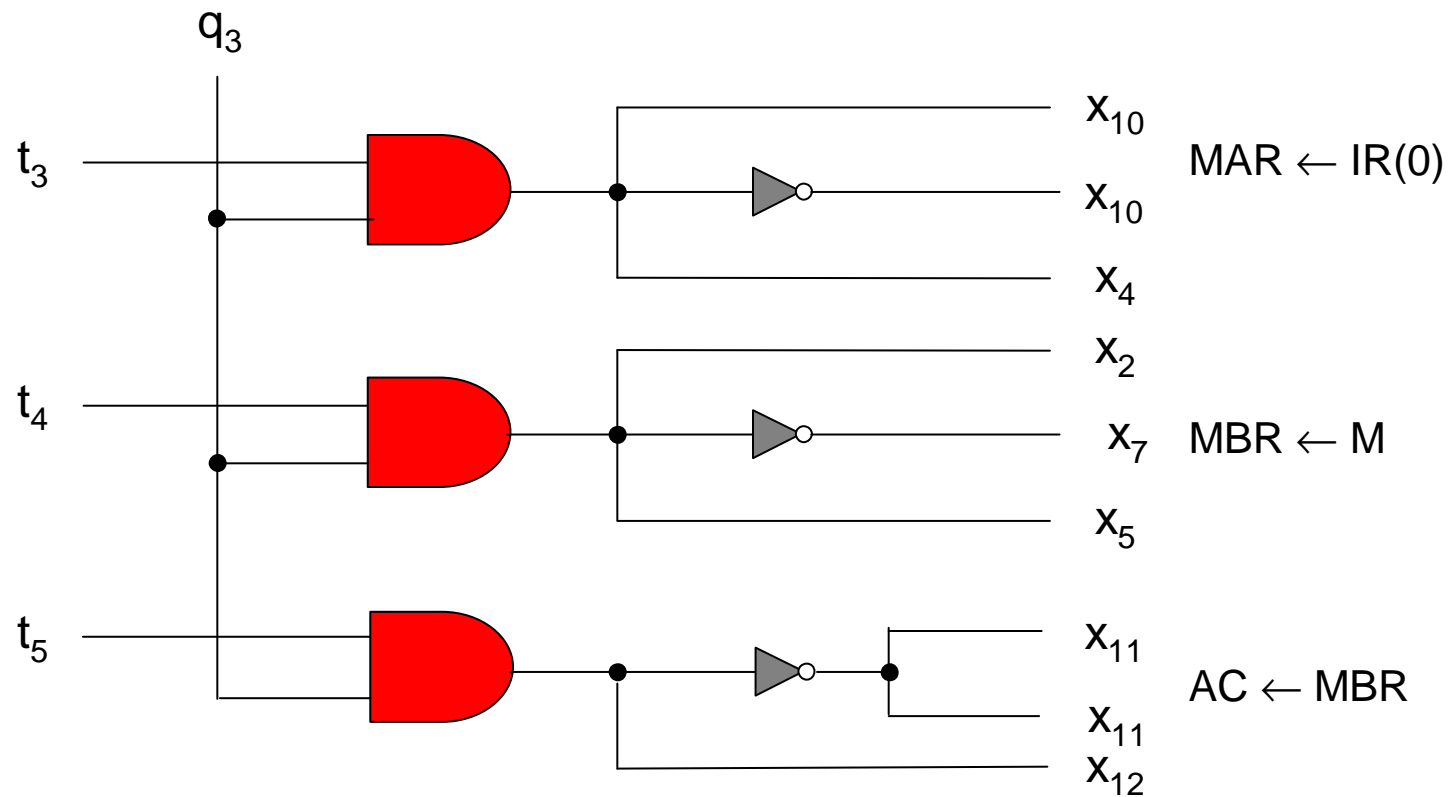
- q_5t_3 : $MAR \leftarrow IR(O)$
- q_5t_4 : $MBR \leftarrow M$
- q_5t_5 : $AD \leftarrow MBR$
- q_5t_6 : $AD \leftarrow AD + AC$
- q_5t_7 : $AC \leftarrow AD$

Logic for the Fetch Cycle

- t_0 : $MAR \leftarrow PC$
- t_1 : $MBR \leftarrow M$; $PC \leftarrow PC + 1$
- t_2 : $IR \leftarrow MBR$



Logic for Loading the Accumulator



CLU Logic

- **Some of the output lines from the two previous slides appear in both circuits. It is necessary to have some logic to connect and coordinate the individual outputs to the wires leaving the CLU.**

Exercises

- **Write microprograms for STA, STI, and JMZ. Implement the microprograms in standard logic.**
- **Design the portion of the CLU that determines the two output lines labeled x_{10} . Input to this circuit will be one or both of the lines previously labeled x_{10} in the individual circuits for LDA, LDI, and the other circuits.**
- **Convert the following program to the equivalent set of binary words, as indicated in this chapter. This is called machine code. Trace the execution of the program by listing the q, t, and x variables.**
 - LDA 1
 - ADD 2
 - STA 3