CS 550 - Programming Languages

Random Access Machines

Jeremy R. Johnson

Introduction

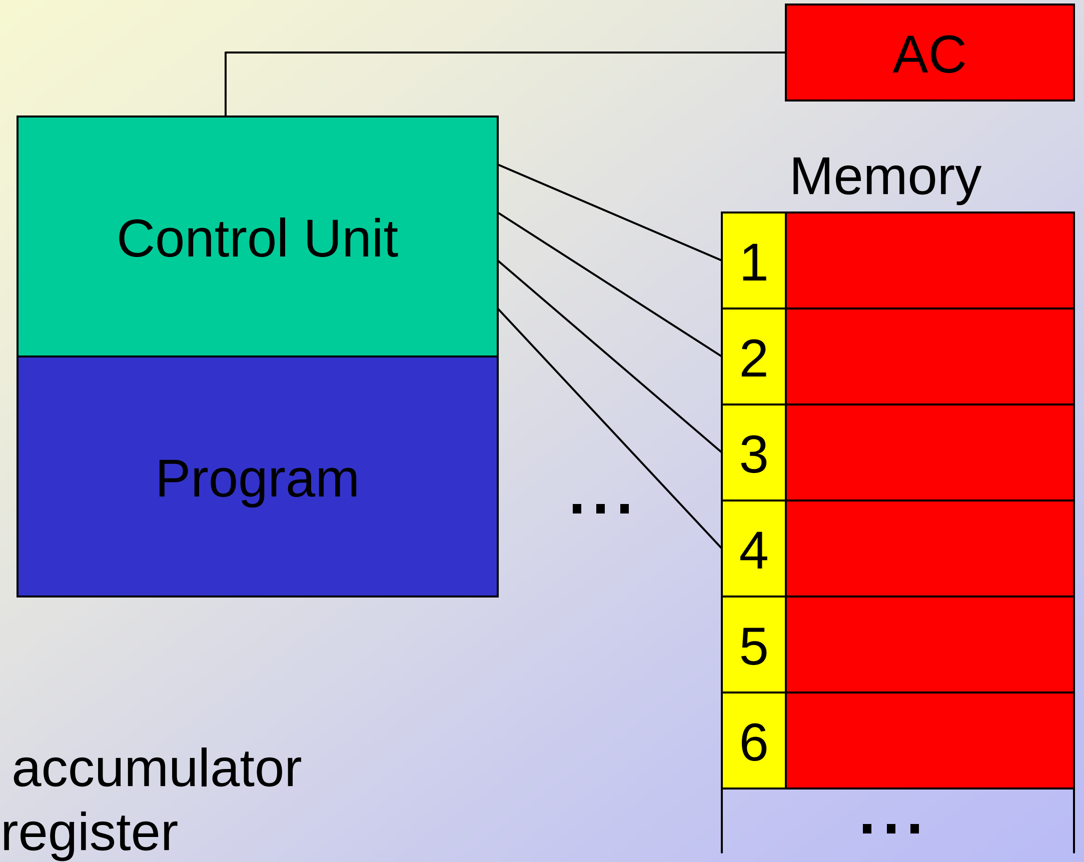
- Objective: To introduce a simple model of a computer that will be used to operationally define the semantics of the Mini Language. In the following lecture, a compiler will be constructed that translates Mini Language Programs to equivalent programs that execute on a RAM using RAM assembly language (RAL).

- 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. In practice the size of the memory is bounded.

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



AC = accumulator register

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** $\leftarrow$ **1** to **n** do

if **B(A(i))** $\neq$ **0**

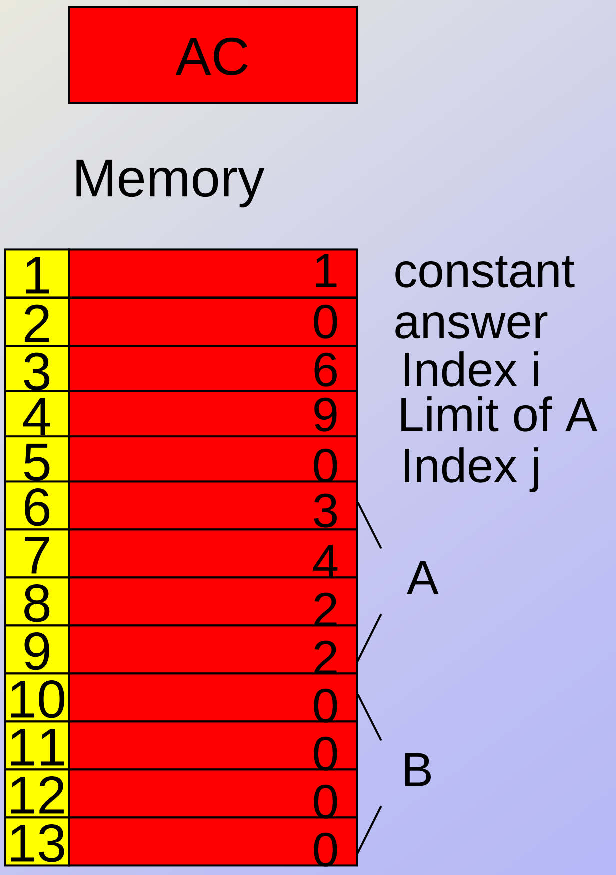
then output A(i);

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;



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 >= 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 ;

AC

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.