Computer Architecture

Prof. Dr. Nizamettin AYDIN

naydin@yildiz.edu.tr

http://www.yildiz.edu.tr/~naydin

1

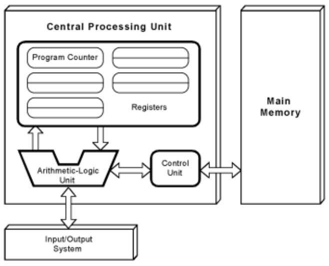
A virtual processor for understanding instruction cycle

The Visible Virtual Machine (VVM)

2

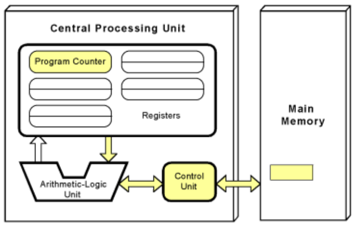
The von Neumann Model

- This is a general depiction of a von Neumann system:
- These computers employ a fetch-decode-execute cycle to run programs as follows . . .



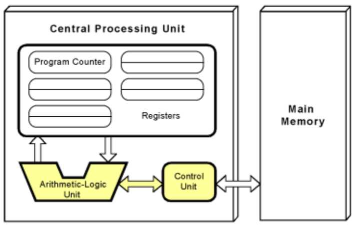
3

- The control unit fetches the next instruction from memory using the program counter to determine where the instruction is located.



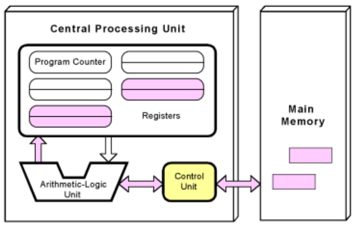
4

- The instruction is decoded into a language that the ALU can understand.



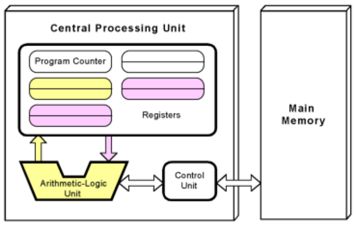
5

- Any data operands required to execute the instruction are fetched from memory and placed into registers within the CPU.



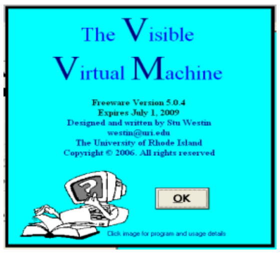
6

- **The ALU executes the instruction and places results in registers or memory.**



7

A virtual processor for understanding instruction cycle


The Visible Virtual Machine

Freeware Version 5.0.4
Expires July 1, 2009
Designed and written by Stu Westin
westin@uri.edu
The University of Rhode Island
Copyright © 2006. All rights reserved

8

The VVM Machine

- The Visible Virtual Machine (VVM) is based on a model of a simple computer device called the Little Man Computer which was originally developed by Stuart Madnick in 1965, and revised in 1979.
- The VVM is a virtual machine because it only appears to be a functioning hardware device.
- In reality, the VVM "hardware" is created through a software simulation. One important simplifying feature of this machine is that it works in decimal rather than in the traditional binary number system.
- Also, the VVM works with only one form of data - decimal integers.

9

Hardware Components of VVM

- **I/O Log**. This represents the system console which shows the details of relevant events in the execution of the program. Examples of events are the program begins, the program aborts, or input or output is generated.
- **Accumulator Register** (Accum). This register holds the values used in arithmetic and logical computations. It also serves as a buffer between input/output and memory. Legitimate values are any integer between -999 and +999. Values outside of this range will cause a fatal VVM Machine error. Non integer values are converted to integers before being loaded into the register.
- **Instruction Cycle Display**. This shows the number of instructions that have been executed since the current program execution began.

10

Hardware Components of VVM

- **Instruction Register** (Instr. Reg.). This register holds the next instruction to be executed. The register is divided into two parts: a one-digit *operation code*, and a two digit *operand*. The Assembly Language mnemonic code for the operation code is displayed below the register.
- **Program Counter Register** (Prog. Ctr.). The two-digit integer value in this register "points" to the next instruction to be fetched from RAM. Most instructions increment this register during the *execute* phase of the instruction cycle. Legitimate values range from 00 to 99. A value beyond this range causes a fatal VVM Machine error.
- **RAM**. The 100 *data-word* Random Access Storage is shown as a matrix of ten rows and ten columns. The two-digit memory addresses increase sequentially across the rows and run from 00 to 99. Each storage location can hold a three-digit integer value between -999 and +999.

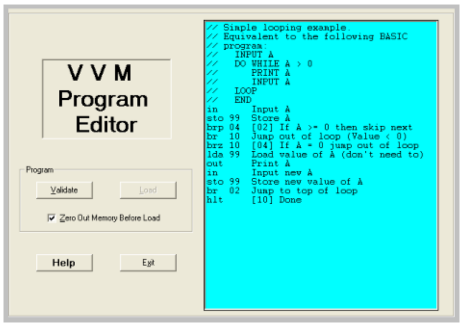
11

Data and Addresses

- All data and address values are maintained as decimal integers.
- The 100 data-word memory is addresses with two-digit addressed in the range 00-99.
- Each memory location holds one data-word which is a decimal integer in the range -999 - +999.
- Data values beyond this range cause a data overflow condition and trigger a VVM system error.

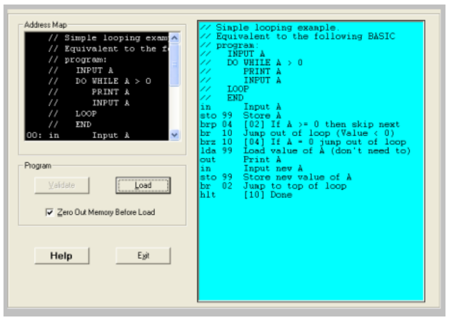
12

VVM Program Editor



```
// Simple looping example.
// Equivalent to the following BASIC
// program:
//    INPUT A
//    DO WHILE A > 0
//       PRINT A
//       INPUT A
//    LOOP
//    END
in       Input A
sto 99   Store A
brp 04   [02] If A >= 0 then skip next
br  10   Jump out of loop (Value < 0)
brz 10   [04] If A = 0 jump out of loop
lda 99   Load value of A (don't need to)
out      Print A
in       Input new A
sto 99   Store new value of A
br  02   Jump to top of loop
hlt      [10] Done
```

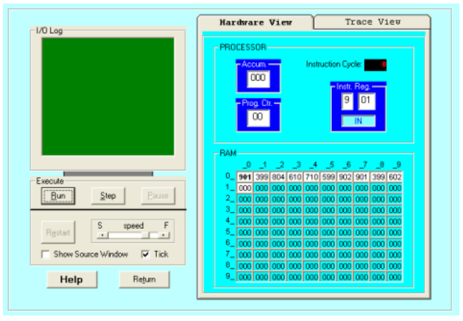
VVM Program Editor



```
    // Simple looping exam
    // Equivalent to the f
    // program:
    //    INPUT A
    //    DO WHILE A > 0
    //       PRINT A
    //       INPUT A
    //    LOOP
    //    END
00: in       Input A
```

```
// Simple looping example.
// Equivalent to the following BASIC
// program:
//    INPUT A
//    DO WHILE A > 0
//       PRINT A
//       INPUT A
//    LOOP
//    END
in       Input A
sto 99   Store A
brp 04   [02] If A >= 0 then skip next
br  10   Jump out of loop (Value < 0)
brz 10   [04] If A = 0 jump out of loop
lda 99   Load value of A (don't need to)
out      Print A
in       Input new A
sto 99   Store new value of A
br  02   Jump to top of loop
hlt      [10] Done
```

VVM Structure and User Interface



Accum.: 000
Prog. Ctr.: 00
Instruction Cycle: 0
Instr. Reg.: 9 01

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9
0_	901	399	804	610	710	599	902	901	399	602
1_	000	000	000	000	000	000	000	000	000	000
2_	000	000	000	000	000	000	000	000	000	000
3_	000	000	000	000	000	000	000	000	000	000
4_	000	000	000	000	000	000	000	000	000	000
5_	000	000	000	000	000	000	000	000	000	000
6_	000	000	000	000	000	000	000	000	000	000
7_	000	000	000	000	000	000	000	000	000	000
8_	000	000	000	000	000	000	000	000	000	000
9_	000	000	000	000	000	000	000	000	000	000

VVM System Errors

- **Data value out of range**. This condition occurs when a data value exceeds the legitimate range -999 - +999. The condition will be detected while the data resides in the *Accumulator Register*. Probable causes are an improper addition or subtraction operation, or invalid user input.
- **Undefined instruction**. This occurs when the machine attempts to execute a three-digit value in the *Instruction Register* which can not be interpreted as a valid instruction code. See the help topic "VVM Language" for valid instruction codes and their meaning. Probable causes of this error are attempting to use a data value as an instruction, an improper *Branch* instruction, or failure to provide a *Halt* instruction in your program.
- **Program counter out of range**. This occurs when the Program Counter Register is incremented beyond the limit of 99. The likely cause is failure to include a *Halt* instruction in your program, or a branch to a high memory address.
- **User cancel**. The user pressed the "Cancel" button during an *Input* or *Output* operation.

The Language Instructions

- **Load Accumulator (5*nn*) [LDA *nn*]** The content of RAM address *nn* is copied to the Accumulator Register, replacing the current content of the register. The content of RAM address *nn* remains unchanged. The Program Counter Register is incremented by one.
- **Store Accumulator (3*nn*) [STO *nn*]** or **[STA *nn*]** The content of the Accumulator Register is copied to RAM address *nn*, replacing the current content of the address. The content of the Accumulator Register remains unchanged. The Program Counter Register is incremented by one.
- **Add (1*nn*) [ADD *nn*]** The content of RAM address *nn* is added to the content of the Accumulator Register, replacing the current content of the register. The content of RAM address *nn* remains unchanged. The Program Counter Register is incremented by one.

The Language Instructions

- **Subtract (2*nn*) [SUB *nn*]** The content of RAM address *nn* is subtracted from the content of the Accumulator Register, replacing the current content of the register. The content of RAM address *nn* remains unchanged. The Program Counter Register is incremented by one.
- **Input (901) [IN]** or **[INP]** A value input by the user is stored in the Accumulator Register, replacing the current content of the register. Note that the two-digit operand does not represent an address in this instruction, but rather specifies the particulars of the I/O operation (see Output). The operand value can be omitted in the Assembly Language format. The Program Counter Register is incremented by one with this instruction.
- **Output (902) [OUT]** or **[PRN]** The content of the Accumulator Register is output to the user. The current content of the register remains unchanged. Note that the two-digit operand does not represent an address in this instruction, but rather specifies the particulars of the I/O operation (see Input). The operand value can be omitted in the Assembly Language format. The Program Counter Register is incremented by one with this instruction.

The Language Instructions

- **Branch if Zero (7*nn*) [BRZ *nn*]** This is a conditional branch instruction. If the value in the Accumulator Register is zero, then the current value of the Program Counter Register is replaced by the operand value *nn* (the result is that the next instruction to be executed will be taken from address *nn* rather than from the next sequential address). Otherwise (Accumulator >< 0), the Program Counter Register is incremented by one (thus the next instruction to be executed will be taken from the next sequential address).
- **Branch if Positive or Zero (8*nn*) [BRP *nn*]** This is a conditional branch instruction. If the value in the Accumulator Register is nonnegative (i.e., >= 0), then the current value of the Program Counter Register is replaced by the operand value *nn* (the result is that the next instruction to be executed will be taken from address *nn* rather than from the next sequential address). Otherwise (Accumulator < 0), the Program Counter Register is incremented by one (thus the next instruction to be executed will be taken from the next sequential address).
- **Branch (6*nn*) [BR *nn*]** or**[BRU *nn*]** or **[JMP *nn*]** This is an unconditional branch instruction. The current value of the Program Counter Register is replaced by the operand value *nn*. The result is that the next instruction to be executed will be taken from address *nn* rather than from the next sequential address. The value of the Program Counter Register is not incremented with this instruction.

The Language Instructions

- **No Operation (4*nn*) [NOP]** or **[NUL]** This instruction does nothing other than increment the Program Counter Register by one. The operand value *nn* is ignored in this instruction and can be omitted in the Assembly Language format. (This instruction is unique to the VVM and is not part of the Little Man Model.)
- **Halt (0*nn*) [HLT]** or **[COB]** Program execution is terminated. The operand value *nn* is ignored in this instruction and can be omitted in the Assembly Language format.

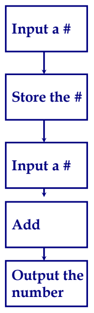
Embedding Data in Programs

- Data values used by a program can be loaded into memory along with the program. In Machine or Assembly Language form simply use the format "*snnn*" where *s* is an optional sign, and *nnn* is the three-digit data value. In Assembly Language, you can specify "DAT *snnn*" for clarity.

Assembly Language

- Specific to a CPU
- 1 to 1 correspondence between assembly language instruction and binary (machine) language instruction
- *Mnemonics* (short character sequence) represent instructions
- Used when programmer needs precise control over hardware, e.g., device drivers

Example - Add 2 Numbers



- Assume data is stored in mailbox address 90
- Let us write instructions (using Mnemonics)

00	IN;	input 1st Number
01	STO 90;	store data in memory location 90
02	IN;	input 2nd Number
03	ADD 90;	add 1st # to 2nd #
04	OUT;	output result
05	COB;	stop
90	DAT 00;	data

Example - Add 2 Numbers (Using Machine code)

Mailbox	Code	Instruction Description
00	901	;input 1st Number
01	390	;store data in memory location 90
02	901	;input 2nd Number
03	190	;add 1st # to 2nd #
04	902	;output result
05	000	;stop
90	000	;data

Program Control

- Branching (executing an instruction out of sequence)
  - Changes the address in the counter
- Halt

	Content	
	Op Code	Operand (address)
BR (Jump)	6	xx
BRZ (Branch on 0)	7	xx
BRP (Branch on +)	8	xx
COB (stop)	0	(ignore)

Instruction Set

ADD	1xx	ADD
SUB	2xx	SUB
STA or STO	3xx	STORE
LDA	5xx	LOAD
BR or BRU or JMP	6xx	JUMP
BRZ	7xx	BRANC ON 0
BRP	8xx	BRANCH ON +
IN or INP	901	INPUT
OUT or PRN	902	OUTPUT
NOP or NUL	400	No Operation
HLT or COB	000	HALT (coffee break)

Example - Find Positive Difference of 2 Numbers

00	IN	901	;input 1st Number
01	STO 10	310	;store data in memory location 10
02	IN	901	;input 2nd Number
03	STO 11	311	;store data in memory location 11
04	SUB 10	210	;subtract 1st # from the 2nd #
05	BRP 08	808	;test
06	LDA 10	510	;if negative, reverse order
07	SUB 11	211	;subtract 2nd # from the 1st #
08	OUT	902	;print result and
09	HLT	000	;stop
10	DAT 00	000	;used for data
11	DAT 00	000	;used for data