Kid Processing Unit Curriculum

A Fun Computer Architecture Introduction

Kid Processing Units: *Begin Here!*

To Get Started Immediately:

1. Print-out block assignments (Page 9 - Page 13), and the desired program example (from Page 15 on)

2. Have the children read their description out loud and discuss their function in the system.

3. Instruct the child playing the CPU to follow the steps in the program until completion.

When Complete:

Check the answer computed! Was there an error? If so, discuss where it might have happened and try again? If not, try running the program again with a timer and see how quickly the steps can be processed.

Next Steps:

Continue this way with each program. Let the children choose or even write their own programs for the next class. Rotate block assignments so each child can experience different functions.

Teach a child to fish…

> *"The acquisition of basic computing skills by any set of children can be achieved through incidental learning provided the learners are given access to a suitable computing facility, with entertaining and motivating content and some minimal (human) guidance."* - Dr. Sugata Mitra, of the Hole-in-the-Wall Education Project (HiWEP).

A personal note from the Author:

Not every child will grow up to write a best selling novel, or paint a masterpiece, or deduce the unifying theoretical framework of physics, but every child can have a chance to discover or even master something new to them, just as inventors and artists of the past did for the very first time.

We all start with addition and subtraction before moving on to geometry and algebra. So why shouldn't kids be able to learn from the same basic building blocks from which Bill Gates and Steve Jobs started their careers?

I've been watching the ongoing "One Laptop Per Child" initiative for developing-world children with some disappointment. Instead of a magic plastic box or smartphone app, at least a "tinkerer adept" minority of kids could really use a more basic system to learn rudimentary computer concepts....on a machine that is so inexpensive that even third-world parents on $2/day aren't tempted sell it for a month's worth of food.

Certainly flashy and cartoonish educational software on full color displays may provide a convenient digital whiteboard to teach math and language. However, in this age of educational computer programs, we experience myriad instances of cashiers who cannot make change without the register doing the math, or adults posting on Twitter who are unable to coherently assemble a complete thought. Children who learn to write with crayons, paint with watercolors or play with Lego building blocks grow into writers, painters, mechanics and engineers who readily adapt their skills to the use of devices and software For a free and fair technological society to flourish, it is critical that kids of all levels have a chance to learn to identify, if not completely master what's under the hood in the structure of computers.

Below is an excerpt from a 2014 email exchange with Lee Hart, developer of classic "retro-computer" kits with the same architecture as the early microcomputers that Steve Wozniak and Paul Allen would have loved to have had as a child:

> *...I well remember the excitement and magic of building and programming my very own computer. That magic is pretty much lost in today's clever little "appliance computers".*

There is a quote I like by Geoffrey Orsak, the dean of engineering at Southern Methodist University: "All children are natural born engineers. Watch them at play. They're not just playing; they're building, learning, experimenting, and creating. They are engineering! Then we put them in school and spend years squashing it out of them."

I think he has it exactly right. We pretend we want students to learn science and engineering, but then teach it in a rigid formulaic way by rote memorization. That sort of education is fine for mechanistic muscle-skills; but useless for learning to think.

The early computers were "thought amplifiers". They encouraged (even required!) you to think and learn and invent. But today, while computers are wonderfully powerful, it is essentially impossible to build or modify them, or to program them yourself for even the simplest things. Everything depends on hardware that someone else built, and software that someone else wrote for you. How it works is all "magic".

Computers have tremendous potential for education! Just read things like "Mindstorms: Children, Computers, And Powerful Ideas" by Seymour Papert, or look at the "hole in the wall" computer learning project in India:

Sincerely,

Lee Hart

--

Obsolete (Ob-so-LETE). Adjective. 1. Something that is simple, reliable, straightforward, readily available, easy to use, and affordable. 2. Not what the salesman wants you to buy.

Not every child needs to be the next Gates, Jobs, Allen or Woz. There are millions of people involved in STEM, who are critical to the development, manufacturing and distribution of technology who are NOT scientists or engineers. But they all need to understand the basic structure and principles of the machines they are helping deliver to the world.

These groups include from start to finish:

-The Design Team which develops chip function

- The Fabrication Team which builds devices from raw materials

- System Designers who integrate boards and boxes

- Manufacturing Teams who build systems

- Programmers who develop software

- Marketing and Sales people who move the hardware

THERE ARE MANY, MANY MORE PEOPLE INVOLVED IN ELECTRONICS THAN JUST ENGINEERS and SCIENTISTS!!! And everyone needs to know how the pieces fit.

This curriculum promises to introduce the basics of computer architecture via hands-on hardware experience, and solves the persistent problem of hardware distribution and high costs – by letting the children be the computer.

I hope this can help provide a first step into the future for children and teachers who might otherwise be obstructed by expense or access.

Jeffrey C. Dunnihoo

Kid Processing Units

Purpose:

Introduce kids to internal computer architecture in a fun way. This is not about how to operate particular computers (Apple, Microsoft..) or program in specific languages (C++, Python) ....rather, to teach about HOW things are going on inside ALL computers with ALL languages, and do it in a very active, engaging way.

We start with a basic introduction to the CPU: Central Processing Unit. But instead...it is a KID PROCESSING UNIT--KPU--where the kids are the circuits.

What we do is explain the function of each block of a basic Von Neumann computer architecture, and then assign a kid or group of kids to perform the task of each of those blocks. These blocks are VERY basic, and the task obviously SIMPLE. The complexity occurs when the "System" starts running and all the blocks (kids) interact.

This can turn into a wild activity, because the kids won't know the OUTPUT of the system until the PROGRAM is completed.

There is a similar application described in a BBC Four video from Professor Dave Cliff that is limited to basic logic gates, but the devices used here in this method allow for higher abstraction of the “block” functions. This way, a program can “store” a number or name in a “memory unit” with the natural understanding of the child about numbers and letters, and without initially introducing complex concepts of binary coding and bytes or variable types.

Imagine we tape out a similar computer block diagram on the floor (Note: optional. If you would prefer not to tape the floor, just give the kids the name tags of their blocks.) and assign one kid to be the Arithmetic Logic Unit, one kid to be the memory controller, maybe a dozen or so kids to be memory registers who must remember the last number they were given, and so on. Each participant has a task and rules.

The control unit is the teacher (or responsible kid). This person asks the memory controller for instructions and then asks the ALU for the answer and then gives the result back to the memory controller who runs over to the correct memory address and whispers the result to that memory location.

So the kids are all running around whispering the answers, and then the program "prints" out the answer by reading out each memory location and writing it on a whiteboard or printer paper.

This abstraction of the block functions is naturally understood by the children, and immediately associates their existing skills with the basic block functions common to all computers.

Programs:

One example would be to have one kid type input on a (dummy) keyboard, and the I/O controller kid would watch the keys pressed and "buffer" or hold that information until the central unit asked for ("reads") it, etc.

The program function might be something as simple as multiplying or adding two large numbers. We run through all the cycles it takes to actually compute it, and then print the result. The answer slowly gets assembled on the whiteboard/output-buffer by the KPU group, but the "Operator" is not allowed to tell the whole group what the input was until the end. Each person only knows their small subset of the problem.

This dramatic interaction provides a fun, approachable introduction into the inner-workings of computers, for students and teachers alike.

Future Extensions:

Perhaps even in the very next class, we can go down one step in more detail to digital logic where each kid is a NAND gate and we run a fast adder in hardware or something like that. Each kid has two inputs (their hands, high/low) and one output (stand or sit). The circuit is wired up with strings tied between kids. ....or...Each kid is a programmable gate. inputs come into the right hand, output is on the left. So an OR-GATE would have two strings tied around the right wrist going off to two other kids' left hands. Each kid gets "programmed" as a logic function. and the operator goes around and programs the inputs on one side of the room, and the circuit generates the result on the other.

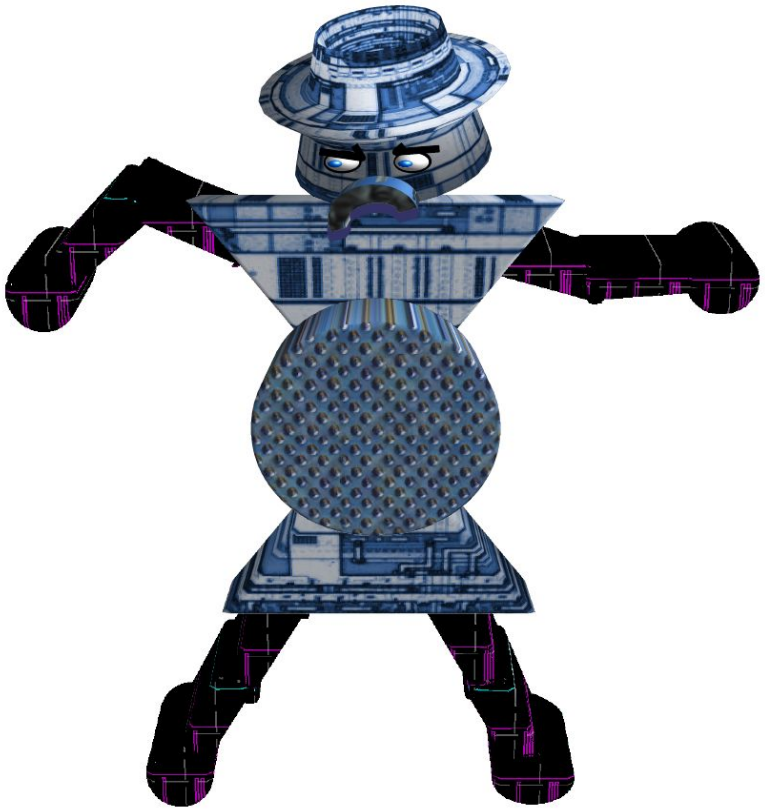
Here the lesson plan shows how to write out the configurations of the kids, the "programs" and such. We could even have the kids take an optional assignment to go home and write their own programs to be run at the next class.

There are many potential levels of teaching that can come out of this. For example. ***Debugging***! run a

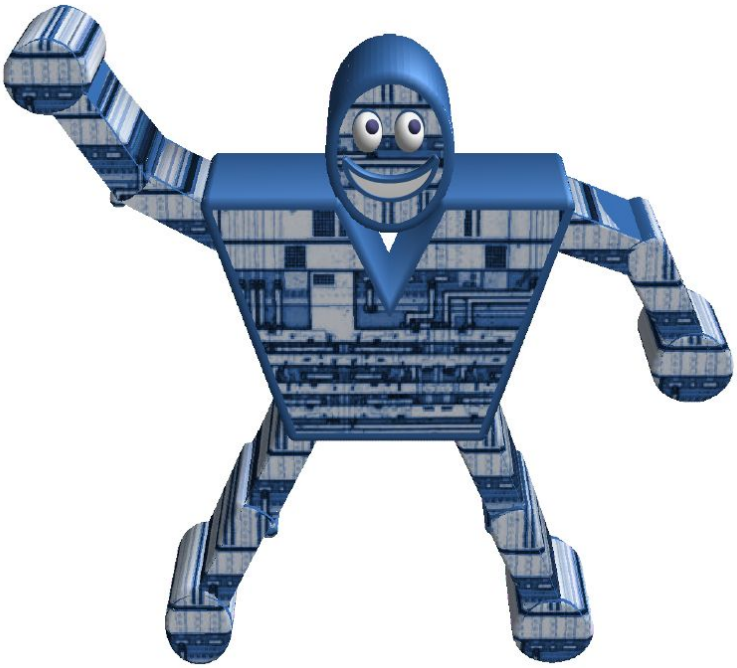
circuit correctly. Then, secretly have one of the kids introduce an error...false logic. Then the group has to look at the output and work backward to find where the bug is. Many many branches possible.

CENTRAL CONTROLLER (CPU)

The Central Controller (aka Central Processing Unit, or “CPU”) reads the program and directs the other Controller Units.



MATH UNIT

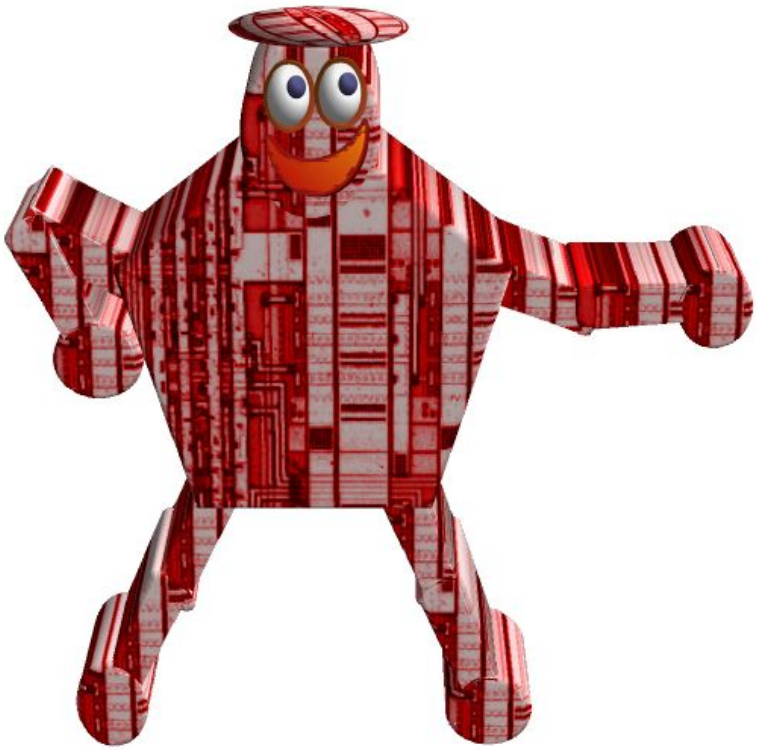


The Math unit (aka Arithmetic Logic Unit, or “ALU”) operates on two values at any given time.

It Adds, Subtracts, Multiplies or Divides, and then returns a result to the Memory Controller and/or the Central Controller.

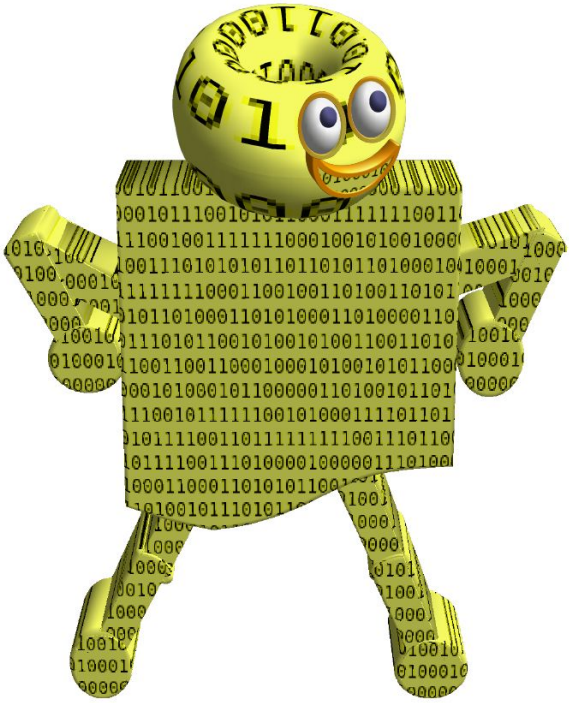
It can also compare values (less than, equal, greater than) so the Central Controller can decide what to do next.

MEMORY CONTROLLER



The Memory Controller assigns and retrieves data from Memory Unit Locations and provides it to the Math Unit and Central Controller as requested.

MEMORY UNIT



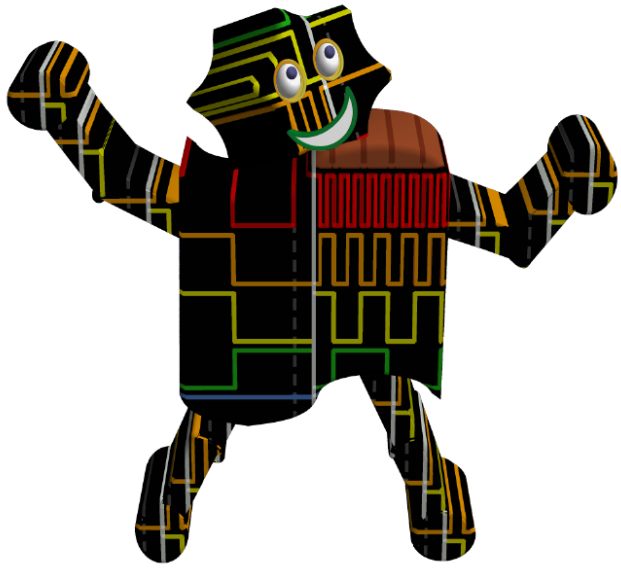
Memory Units can be named and assigned and partitioned however the Memory Controller Chooses.

In more complex programs, the name “A” may refer to two locations (two children) which hold a “1” and a “5” and the Memory controller knows how to report this as “15” to the Central Controller.

The organization of the Memory Units, ordering of the locations and communication of values between them and the rest of the computer is determined solely by the Memory Controller.

Memory Allocation and Management changes. It is “*dynamic*.”

INPUT / OUTPUT CONTROLLER (IOCTL)

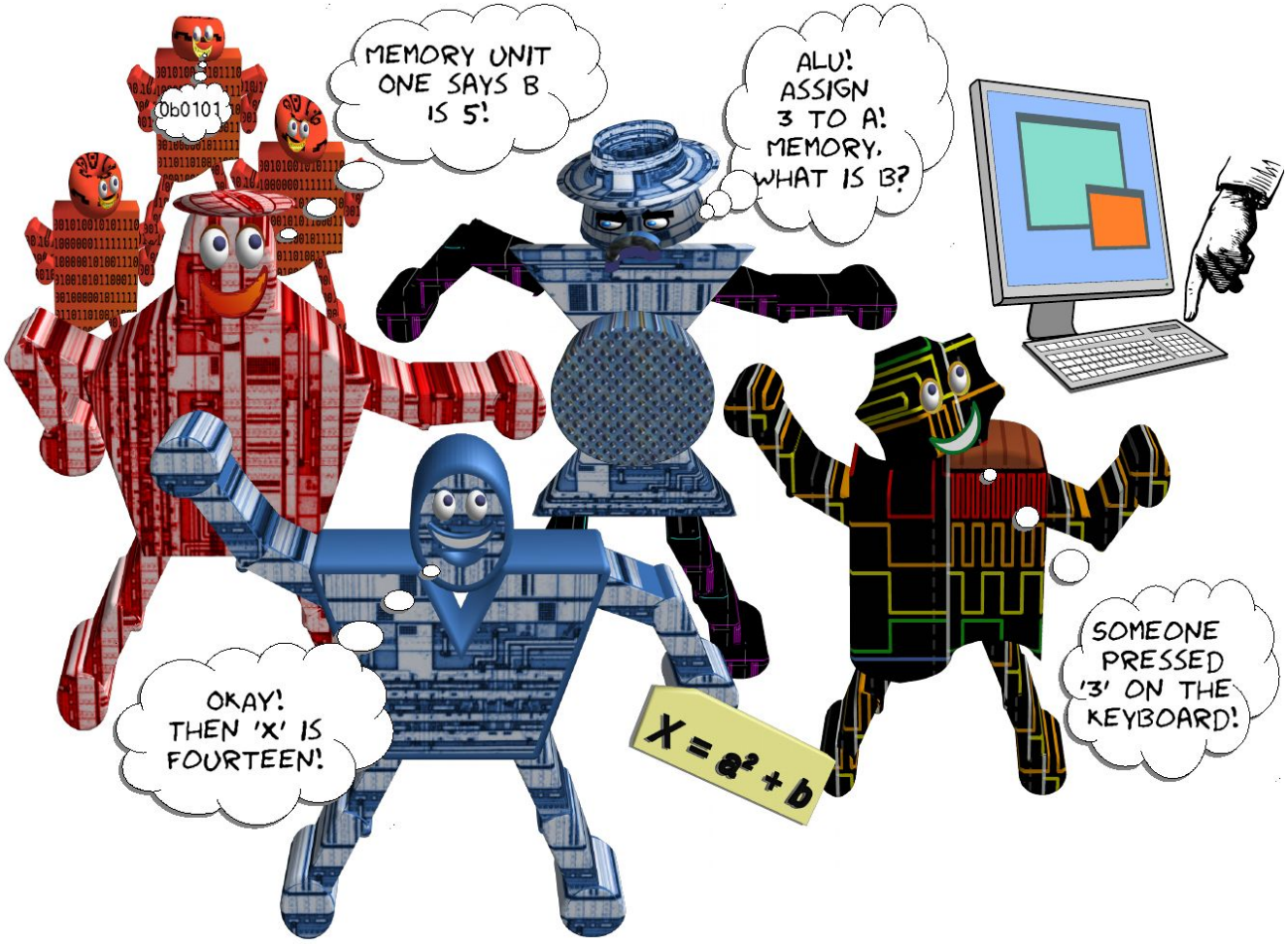


An Input Controller relays User Input to the Central Controller (Keyboards, Mice...etc)

An Output Controller presents Computer Output to the User (Display, Printer, etc)

Program Operation

When a program is “running” there may be multiple operations happening at the same time, and seemingly chaotic confusion. Interestingly, though, as long as each person keeps to their part, communicates only to those assigned, and waits for the responses as required, out of apparent chaos will emerge a simple and straightforward result.



PROGRAM EXAMPLES

PROGRAM 1: ADD THREE NUMBERS

This program uses a group of eight students as a computer to add three numbers together and output the result.
Kids Required:
1 CPU
1 Math Unit
1 Memory Controller
1 I/O Controller
3 Memory Units
1 Computer Operator

Although this is a very simplistic exercise, it demonstrates how the complex and rigid organization of the individual blocks (kids) within the system creates the ability to perform tasks as a whole which might even be more complicated than the individual child may be capable of (only adding three numbers here, but consider that a real computer essentially does the complex functions by merely doing billions of these operations very quickly).

STEP 1> READ INPUT into Memory [A]

Here the Input Controller asks the "operator" for a number (any number that is appropriate for the children's age, e.g. integers from 1 to 5, or fractions or decimals for older students.) The Input Controller tells the CPU, who then tells the Memory Controller. The Memory Controller decides which Memory Unit is Location "A" and tells them to remember the number. The Memory Controller may decide to assign names to each location before beginning ("initialization") or may choose to select them as each new value is set in each step. Once assigned, the Locations hold their Value and Name until the Memory Controller changes them.

STEP 2> READ INPUT => Memory [B]

Repeat as in #1 for the second location.

STEP 3> READ INPUT => Memory [C]

Repeat again for third location.

STEP 4> READ MEMORY [A] and [B] to the Math Unit

The CPU tells the Memory Controller to ask for Location A's value (memorized from Step 1) and turns to the Math Unit and repeats it. The CPU repeats this request to the Memory Controller for Location B's value (from Step 2) and again tells the Math Unit.

STEP 5> ADD ==> Memory [D]

CPU tells the Math Unit to add the two values just given, and reports the answer to the Memory Controller to be stored in Location D. Memory Controller repeats the value to be memorized to Location D.

STEP 6> READ MEMORY [C] and [D] to the Math Unit

CPU tells the Memory Controller to ask for Location C's value (memorized from Step 3) and turns to the Math Unit and repeats it. Memory Controller repeats this for Location D's value (from Step 5).

STEP 7> ADD ==> Memory [D]

CPU instructs the Math Unit two ADD the two values just given together, and reports the answer from the Math Unit to the Memory Controller to be stored back again in Location D. Memory Controller repeats the value to be memorized to Location D.

STEP 8> WRITE OUTPUT from Memory [D]

The CPU asks the Memory Controller for the value from Location D. The Memory Controller tells the CPU, and the CPU tells the I/O controller, and the I/O controller tells the Computer User.

Questions: Ask the Computer User if the answer is correct. How many steps did it take? Run the program again but change the operation to multiplication.

PROGRAM 2: DECODER

This program uses a group of 12 students as a computer to decode a word and output the result.

Kids Required:
1 CPU
1 Memory Controller
1 Network Controller
1 Display Controller
6 Memory Units [1-6]
1 Remote Computer

This is a larger and more complex program which demonstrates how two computers communicate with numbers. The CPU

REMOTE COMPUTER SENDS CODE: 14,34,14,33,53

STEP 1> WRITE THE VALUE “5” into Memory 6.

Memory 6 will be used as a Memory Index to point the memory controller to different positions.

Display controller is instructed to get ready and keep a cover sheet on the transparency during the program.

STEP 2> READ FIRST INPUT into Memory AT Index [6]

CPU asks I/O Controller for next number input. Memory controller asks Memory #6 the value (at first it is 5, so this points to Memory 5). CPU tells Memory controller to write the Input number (example “14”, to Memory 5.

STEP 3> CPU instructs Memory Controller to Decrement Index [6]

The CPU can only talk to the memory controller. It cannot directly read memory. Memory controller reports “5” to CPU and CPU subtracts 1 and tells Memory Controller to write back “4”.

STEP 4 LOOP> READ NEXT INPUT into Memory AT Index [6] and Decrement

CPU asks I/O Controller for next number input. Memory controller asks Memory #6 the value (now it’s Memory 4). CPU tells Memory controller to write the Input number (example “34”, to Memory 4. Memory controller reports “4” to CPU and CPU subtracts 1 and tells Memory Controller to write back “3”.

STEP 5 LOOP> READ NEXT INPUT into Memory AT Index [6] and Decrement

CPU asks I/O Controller for next number input. Memory controller asks Memory #6 the value (now it's Memory 3). CPU tells Memory controller to write the Input number (example "14", to Memory 3 Memory controller reports "3" to CPU and CPU subtracts 1 and tells Memory Controller to write back "2".

STEP 6 LOOP> READ NEXT INPUT into Memory AT Index [6] and Decrement

CPU asks I/O Controller for next number input. Memory controller asks Memory #6 the value (now it's Memory 2). CPU tells Memory controller to write the Input number (example "33", to Memory 2. Memory controller reports "2" to CPU and CPU subtracts 1 and tells Memory Controller to write back "1".

STEP 7 LOOP> READ NEXT INPUT into Memory AT Index [6] and Decrement

CPU asks I/O Controller for next number input. Memory controller asks Memory #6 the value (now it's Memory 1). CPU tells Memory controller to write the Input number (example "53", to Memory 1. Memory controller reports "2" to CPU and CPU subtracts 1 and tells Memory Controller to write back "0".

STEP 8> READ NEXT INPUT into Memory AT Index [6] and Decrement

CPU asks I/O Controller for next number input. But Remote Computer does not have any more input. So CPU exits loop and begins Decode operation.

NOTE: At this point we have read a serial code into 5 locations in memory, in reverse order as received.

STEP 9> WRITE THE VALUE "5" into Memory 6.

Memory 6 will again be used as a Memory Index to point the memory controller to the data, starting at 5 again.

STEP 10> DECODE Memory AT Index [6] and Decrement

CPU asks for the value pointed by Memory 6 (now "5") and Decodes it using the lookup table below. CPU takes that decoded letter and tells the display controller to write it on the clear "display." The first digit is the row, the second digit is the column. (here, the value at Memory 5 = 14. So row 1

and column 4 = “M”) Display controller writes normally in large letters from left to right on the transparency). CPU tells Memory controller to decrement Memory 6.

STEP 11 LOOP> DECODE Memory AT Index [6] and Decrement

CPU asks for the value pointed by Memory 6 (now “4”) and Decodes it. CPU takes that decoded letter and tells the display controller to write it on the clear “display.” (here, the value at Memory 4 = 34. So row 3 and column 4 = “O”) Display controller writes normally in large letters from left to right on the transparency). CPU tells Memory controller to decrement Memory 6.

STEP 12 LOOP> DECODE Memory AT Index [6] and Decrement

CPU asks for the value pointed by Memory 6 (now “3”) and Decodes it. CPU takes that decoded letter and tells the display controller to write it on the clear “display.” (here, the value at Memory 3 = 14 again. Row 1 and column 4 = “M”) Display controller writes normally in large letters from left to right on the transparency. CPU tells Memory controller to decrement Memory 6.

STEP 13 LOOP> DECODE Memory AT Index [6] and Decrement

CPU asks for the value pointed by Memory 6 (now “2”) and Decodes it. CPU takes that decoded letter and tells the display controller to write it on the clear “display.” (here, the value at Memory 2 = 33. Row 3 and column 3 = “I”) Display controller writes normally in large letters from left to right on the transparency. CPU tells Memory controller to decrement Memory 6.

STEP 14 LOOP> DECODE Memory AT Index [6] and Decrement

CPU asks for the value pointed by Memory 6 (now “1”) and Decodes it. CPU takes that decoded letter and tells the display controller to write it on the clear “display.” (here, the value at Memory 1 = 53. Row 5 and column 3 = “I”) Display controller writes normally in large letters from left to right on the transparency. CPU tells Memory controller to decrement Memory 6.

STEP 13 LOOP> Terminate Loop and Show Result!!!

CPU asks for the value pointed by Memory 6 (now “0”) and halts loop. CPU tells Display Controller to hold the piece of paper on the transparency and hold up the stack to show the class. (transparency in front, and white paper behind as background.)

The Display should have written: “ M O M I H “

(Or, “HI MOM” when viewed by class!)

#2 > #1V	1	2	3	4	5	6
1	F	Q	J	M	S	X
2	*	B	Y	D	K	L
3	A	E	I	O	U	?
4	G	#	C	!	Z	♥
5	T	R	H	P	&	N
6	W	U	★	♦	E	V

SAMPLE CODE: 1434143353

OTHER PROGRAMS

TBD