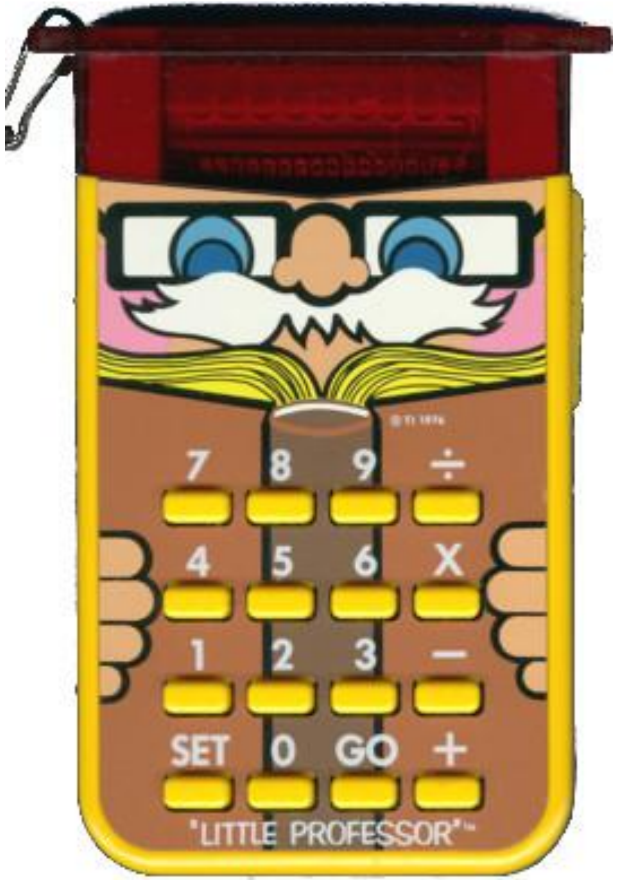
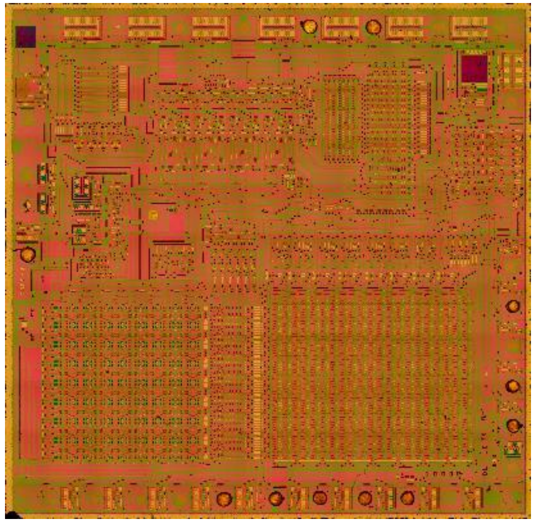


The big adventure of little professor and its 4-bits handheld friends running TMS 1000



PONSARD Christophe
NAM-IP Computer Museum

FOSDEM 24 – Retrocomputing – February 4



Context – NAM-IP Computer Museum

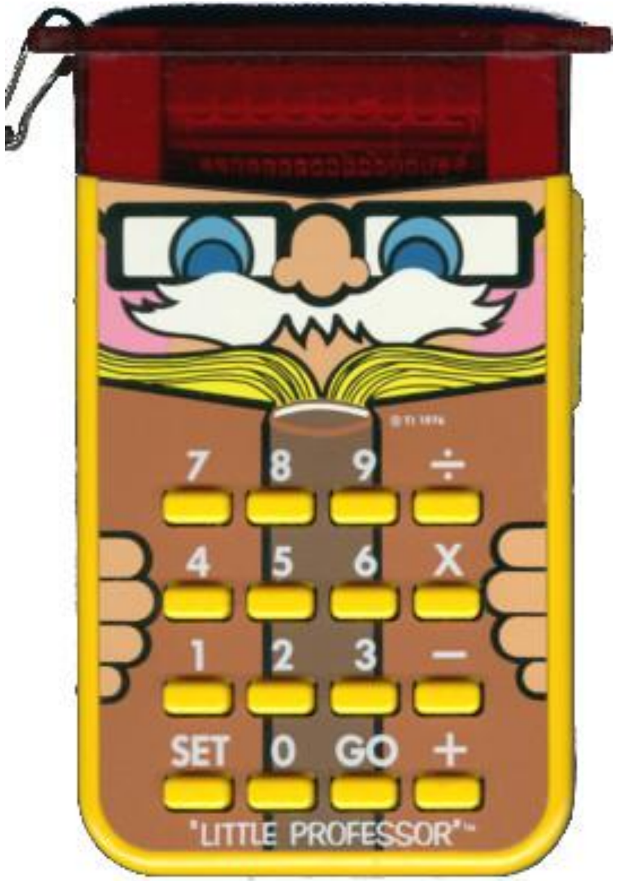
Computer Museum NAM-IP

www.nam-ip.be

- Located in Namur/Belgium - 30’ from Brussels
  - worth a visit if you are staying a few days I Belgium after FOSDEM)
  - also: Pixel Museum (in BXL) & HomeComputerMuseum (in Eindhoven/NL not so far)
- Missions:
  - Preservation: safeguarding digital heritage, focus on local pioneers
  - Acquisition of artefacts, enriching collections: Bull, Burroughs/Unysis, I&B,…
  - Exhibitions: for all, specific animation, permanent/temporary
  - **Research: about machines, software, communities ➔ here BULL TMS1K**
- “Container design”, an historical parallel

How it started by a donation

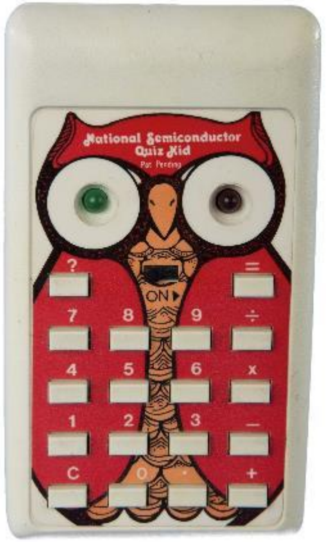
"Little Professor" (v1978)



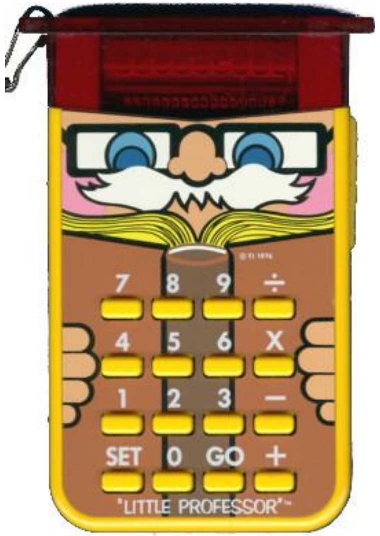
What is it ?

- Launched in 1976 by Texas Instruments ($20)
- "Inverted" calculator = "learning aid" (5-9 yo)
- Generate problems + - * / (4 levels)
- 10 problems, 3 trials before showing answer
  (later versions reduced to 5 + more "rewards")
- Iconic look "wise and friendly owl"
- Huge success
  - ➔many variants/successors by TI
  - ➔In "collective memory" (and museums – here HNF)

Some variants and versions



1975
OWL predecessor
(nat. semicond.)
no generator



1976-1978-1982-now
problems
+ companion game

later: tables

1977 wiz-a-tron
checker
no generator

1977 DataMan
tables, problems
guess, box
2P: force out, orbit

1980 Math Marvel
problems, tables
speed, zap, check
guess

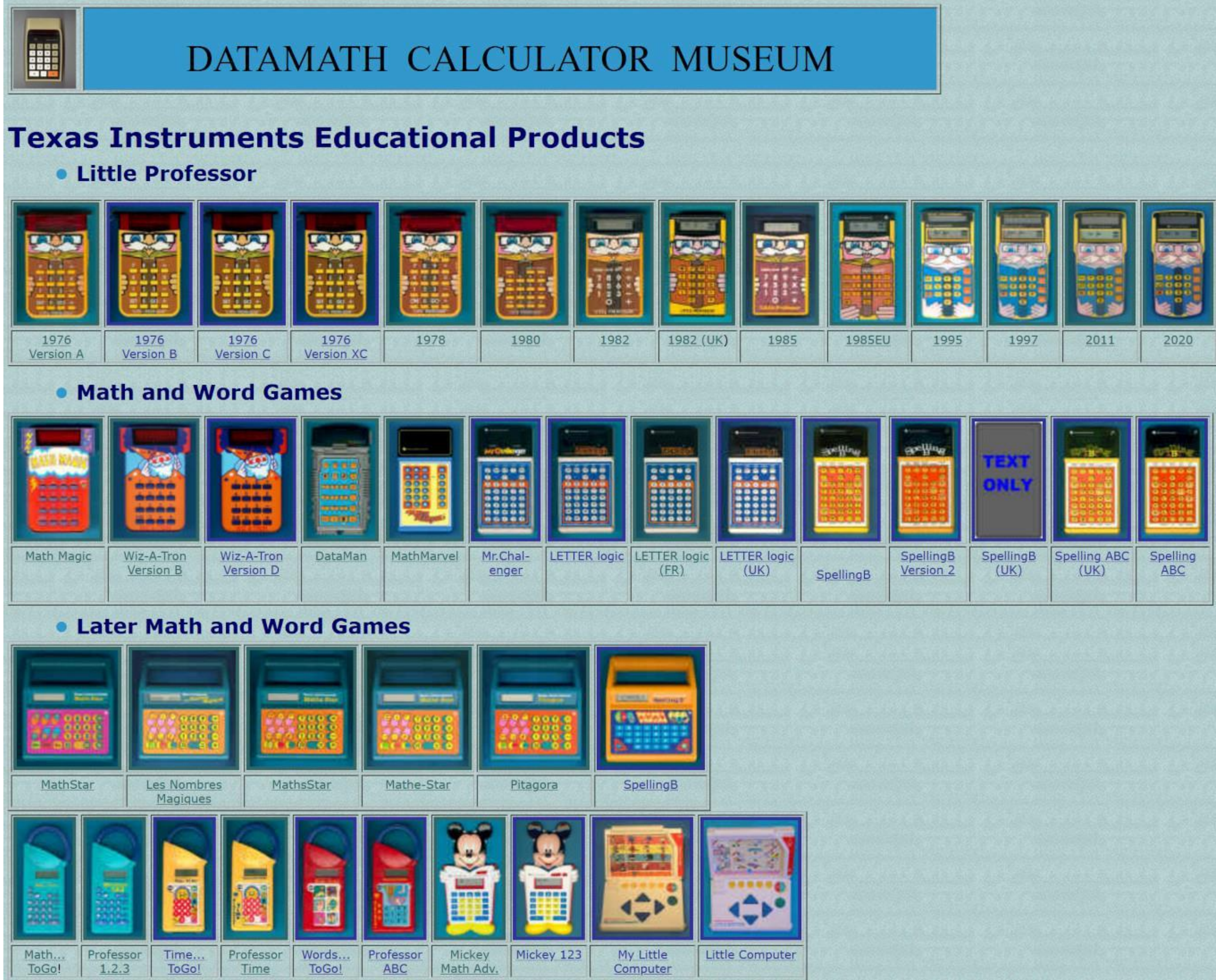
1989 Prof 123
problems/box, tables

Also calculator !

Versions
Variants

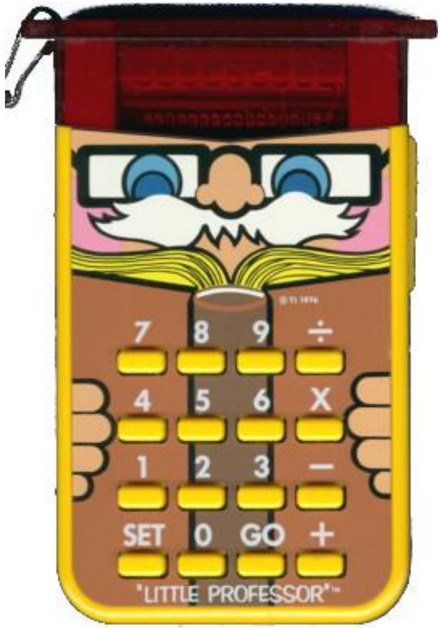
See
http://www.datamath.org

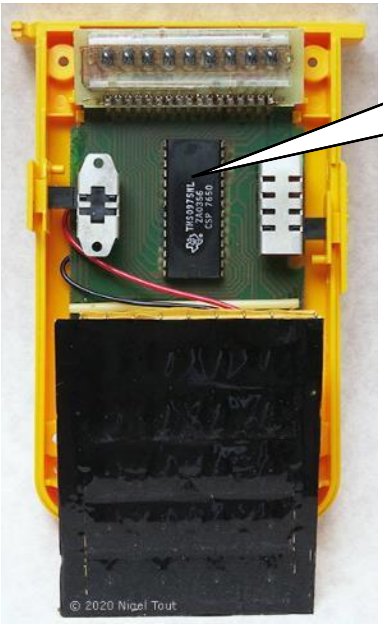
Huge work by
Joerg Woerner !

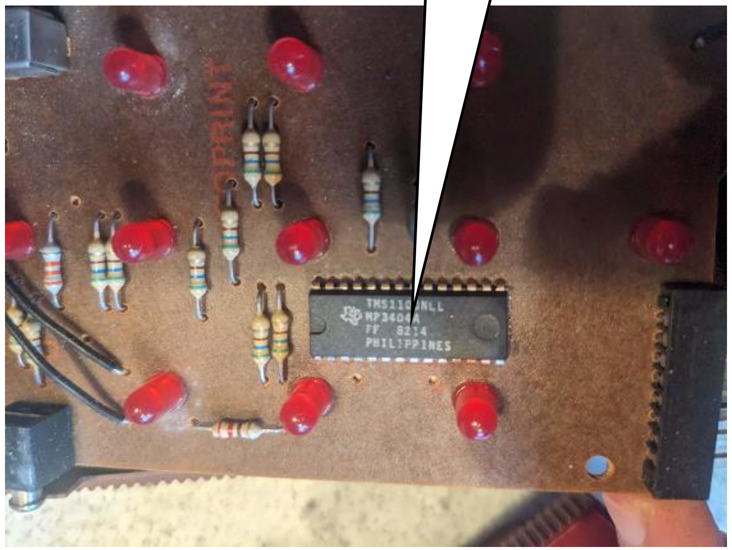


Looking inside – what’s under the hood ?

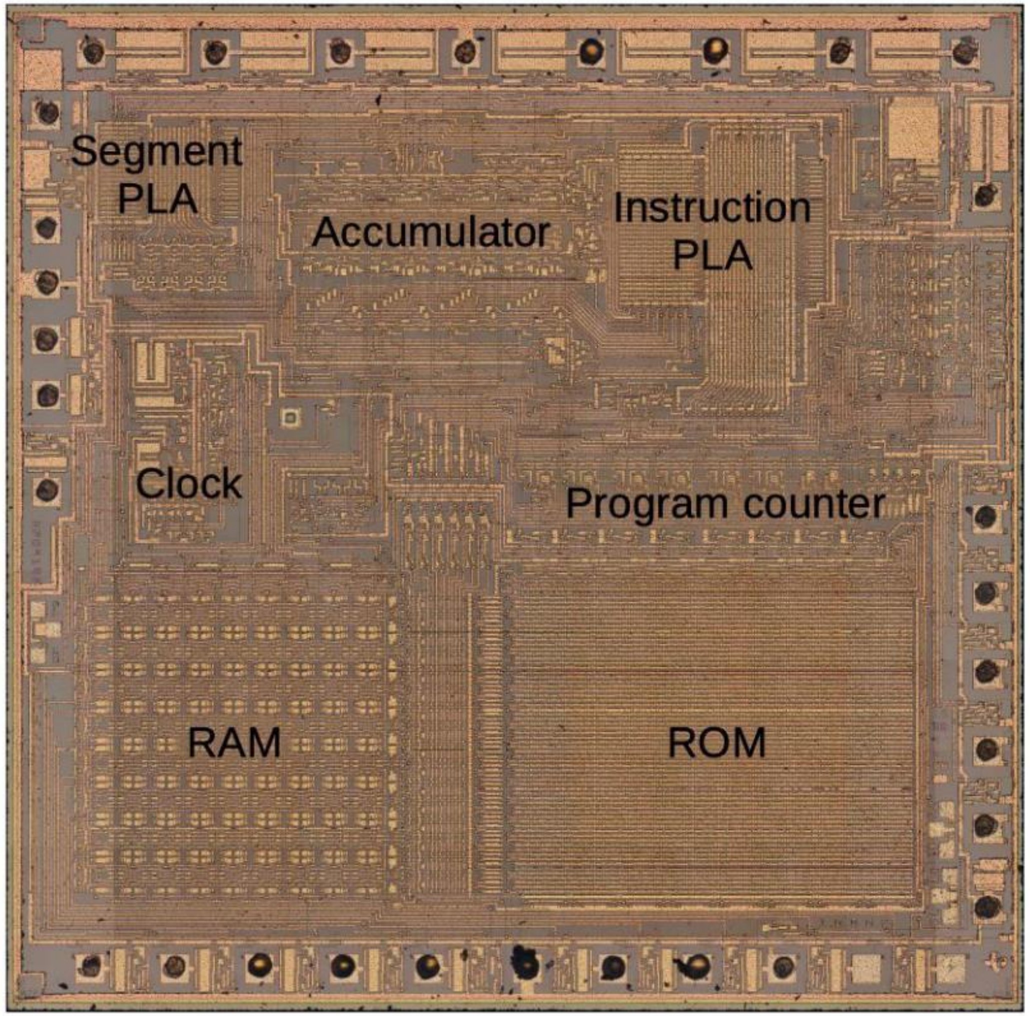
- How does it work ? How is low cost / product lines achieved ?
- ANSWER:
  - same technology as in calculators ☺
  - few components ➔ single chip calculator = **<u>microcontroller</u>** (not µP) !
  - system is typically reduced to single chip + display/keyboard

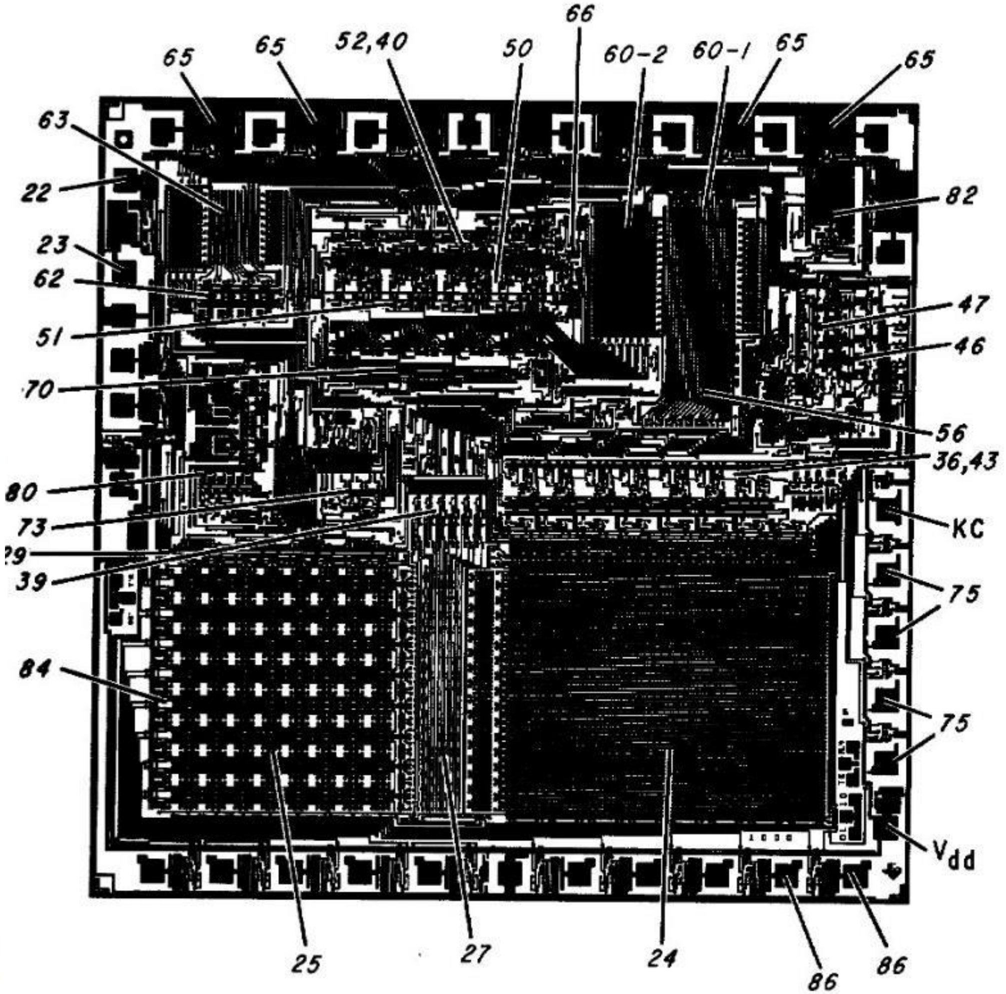






A look at the die





- 65 - output buffers
- 52 - accumulator
- 40 - RAM Y register
- 50 - adder
- 66 - status logic
- 60 - instruction PLA
- 82 - power up clear circuit
- 47 - address buffer
- 46 - page address buffer
- 56 - CKB logic
- 36 - program counter
- 43 - subroutine register
- 75 - keyboard input
- 86 - output buffers
- 24 - ROM
- 27 - ROM/RAM word address decoder
- 25 - RAM
- 84 - output register
- 39 - data select
- 29 - RAM page decoder
- 73 - RAM page address register
- 80 - clock generator
- 70 - RAM write control
- 51 - adder input select
- 62 - output register
- 22 - oscillator input pin
- 23 - oscillator output pin
- 63 - segment decoder PLA

From RAM – see http://www.righto.com/2020/11 - including move from PMOS to CMOS - great analysis by Ken Shirriff

Discovering the TMS "1000" family

- a family of microcontrollers introduced by TI in 1974
  - actually not first generation, so already experienced !
- "computer on chip" combining
  - a 4-bit central processor unit,
  - read-only memory (ROM)
  - random access memory (RAM)
  - input/output (I/O) lines
- Note:
  - Need custom die for each ROM but also provides protection)
  - CORE design : TMS
    ➔ customer version : TMC
  - Harvard architecture >< von Neumann

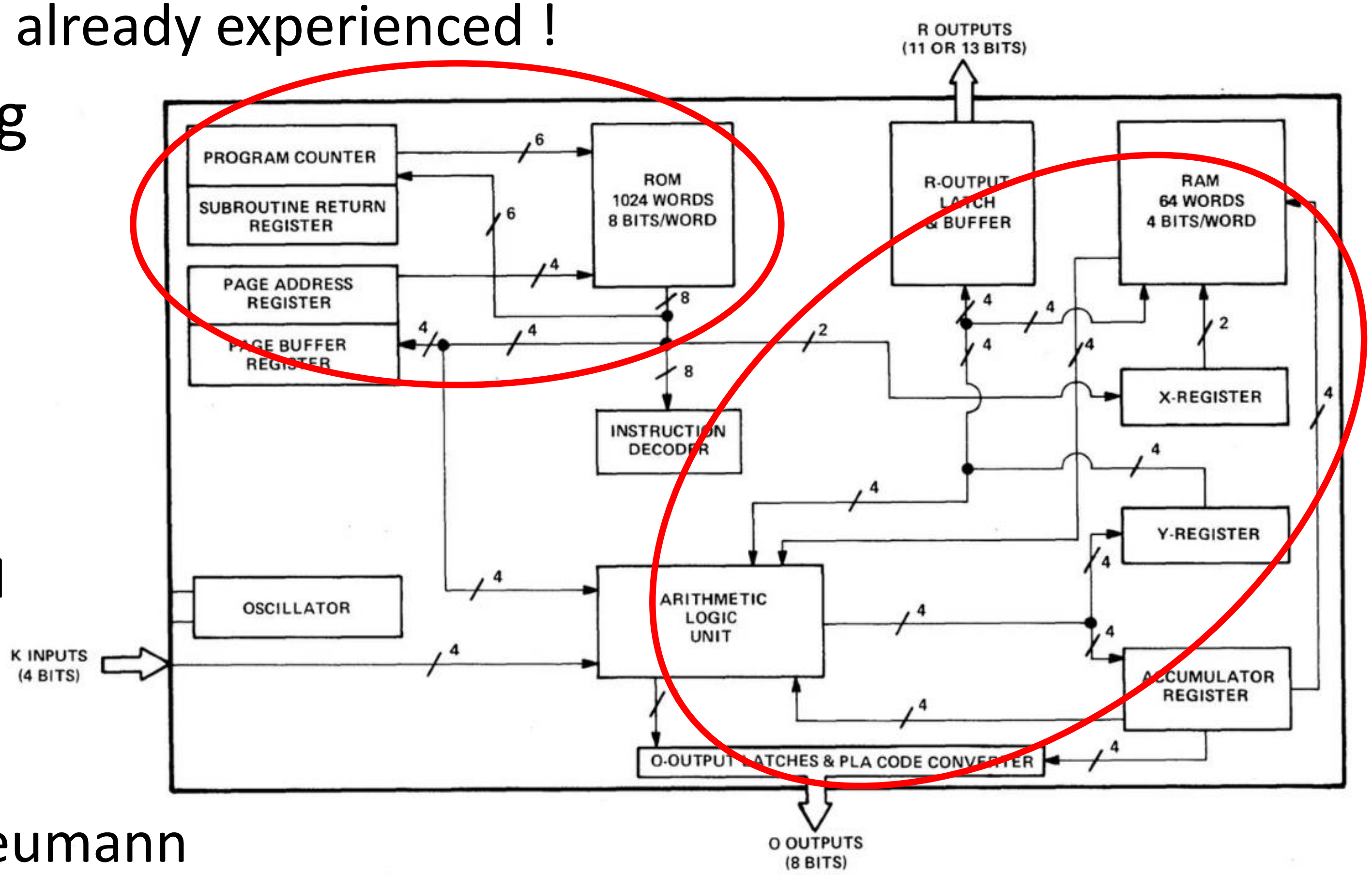
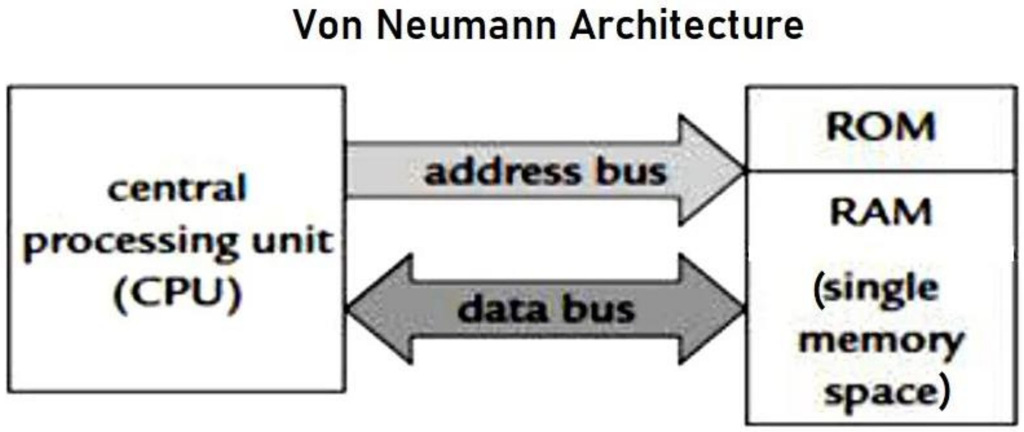
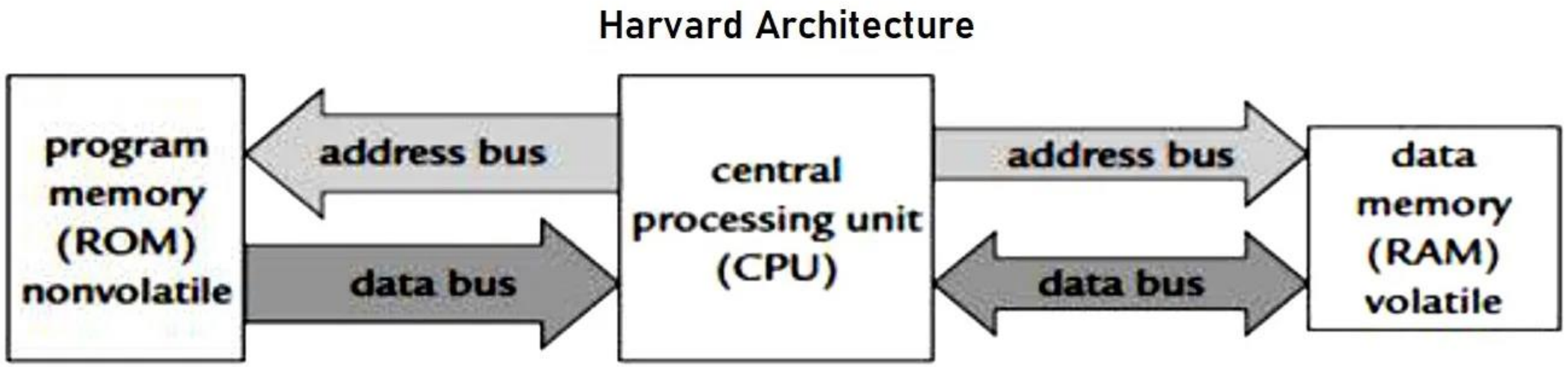


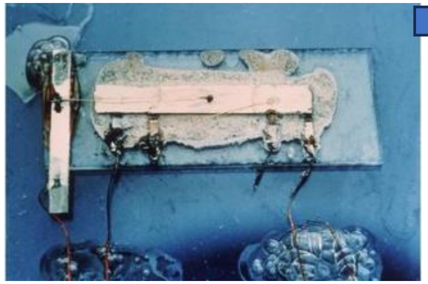
FIGURE 3 – TMS 1000/1200 LOGIC BLOCKS

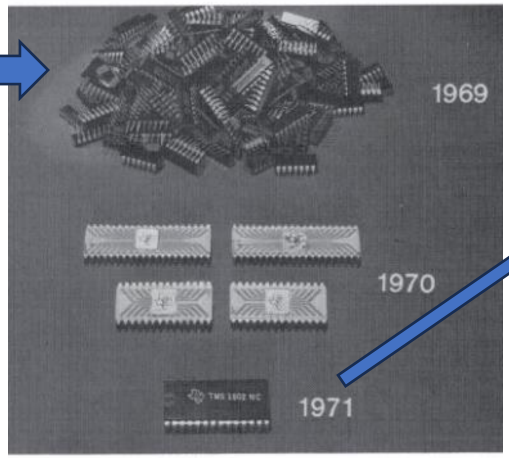
Harvard vs Von Neumann architecture (reminder)





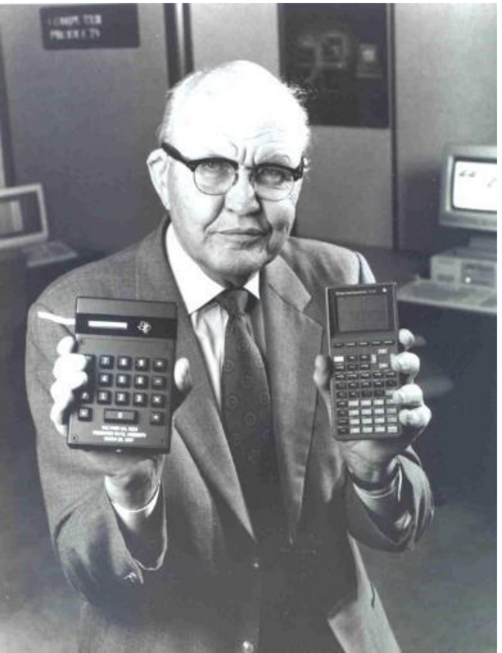
➔ Pending question: how to dump ROM ?? (we need it for emulation !)

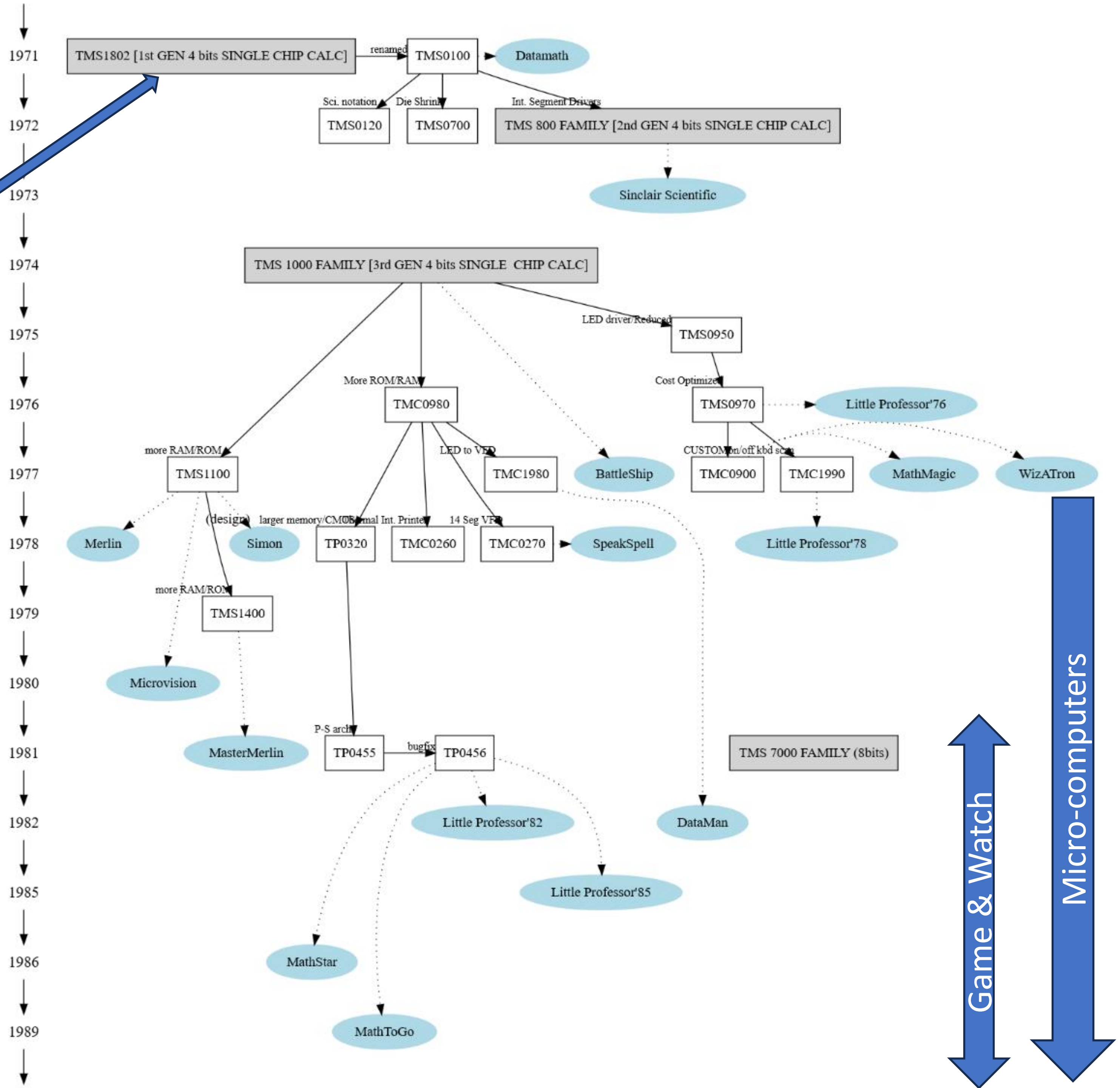




First IC
Kilby 1958 (GE - TI)
&Noyce (SI ➔Intel)

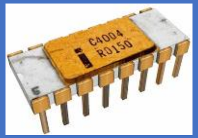
Timeline



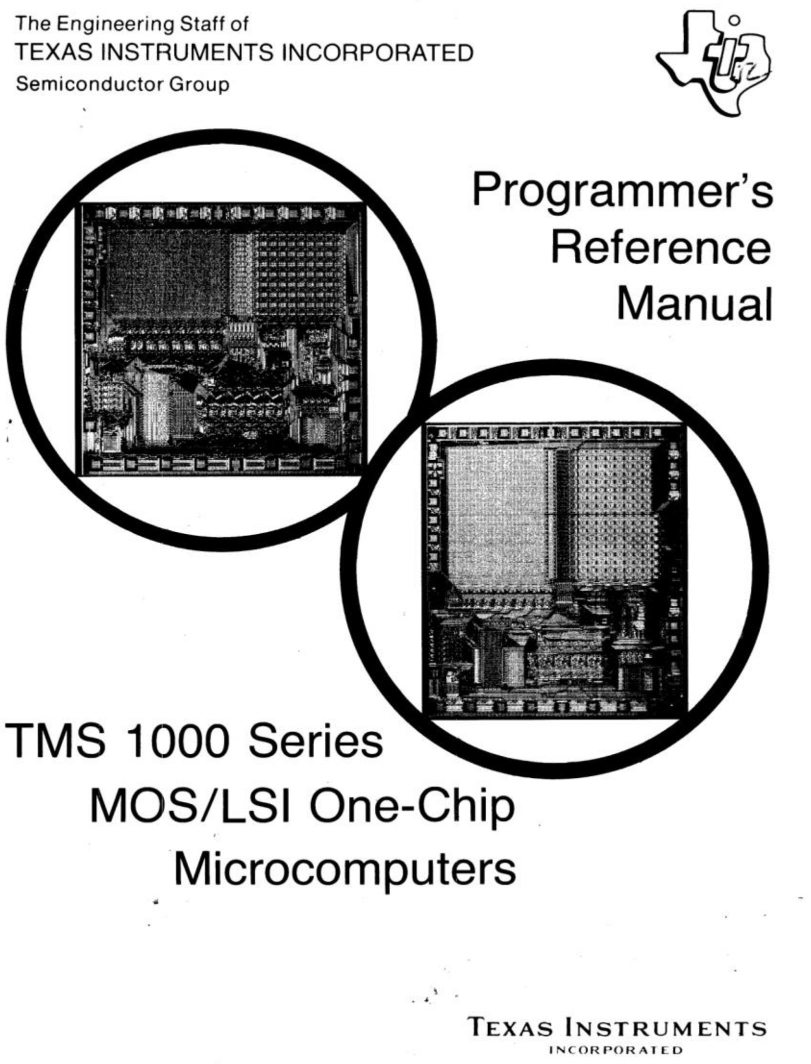


Handheld games not core market: many calculators !

Comparison with 4004 vs TMS 1000

	Intel 4004 	**TMS 1000 family** 
Year	1971-1981	1974-(1989)
Transistors	2300	4000
Freq	750 KHz	200-450 KHz
Price	$60	**$2-$4**
Sales	About 1 million in total	**Millions/year**
Type	Microprocessor: DEC, REG, ALU ➔ Complex integration	Microcontroller: DEC, REG, ALU, RAM+ROM, CLOCK, IO ➔ Single chip
Architecture	Von Neumann	**Harvard**
Bus	4 bits (external/internal) data + address	4 bits (internal)
Instruction set	46 (mostly 8 bits), BCD oriented	43 (base, 8 bits) BCD arithmetic, no logical/shift
Registers	16 (nibbles)	2.5: accumulator + X-Y pointer to RAM (used as registers)
RAM	~1024 nibbles (max 4500 bytes)	64-128 nibbles
ROM	Typically 4K	1K-4K
Applications	pinball machines, traffic light controllers, cash registers, bank teller terminals, blood analyzers, gas station monitors	Many calculators, digital clocks, handheld games, printer controllers, data terminals, appliance control, automotive applications

Programming TMS 1000



- manual from 1975 (232 pages)
  - architecture
  - instruction set, addressing
  - micro-code
  - example of routines, memory mngt
- Other interesting documents: patents !
  - Very well documented
  - Sometimes with ROM dump in source

TMS 1000 Registers, Memory and Flags

TMS1000 registers

0_9 0_8 0_7 0_6 0_5 0_4 0_3 0_2 0_1 0_0 *(bit position)*

		A	accumulator
	X	Y	mem pointer
page add	program ctr		next instruction
page buff	subroutine ret		long branch/return

Branch status flags

C	S	call latch/branch

Note: memory paginated

Need to manage page switching
for long branch/call (using LDP + buff)

Question: where is the stack ?

FILE ADDRESS	REGISTER	Y-REGISTER ADDRESS 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
X = 00	D	OV 0	MSD 9	8	7	6	5	4	3	LSD 2							
X = 01	E	OV 0	MSD 1	2	3	4	5	6	7	8	9	0	1	2	3	4	LSD 5
X = 10	F	OV 0	MSD 5	4	3	2	1	0	9	8	7	6	5	4	3	2	LSD 1
X = 11	G	OV 0	MSD 8	7	6	5	4	3	2	LSD 1							

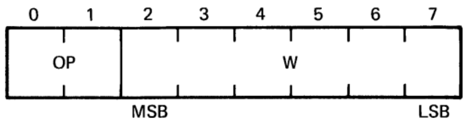
Memory typically used as real “registers”

Tabular addressing 4x16 nibbles

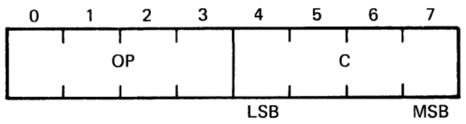
(X,Y) used to point on memory

Instruction set

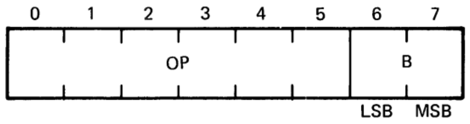
- 43 instructions (base)
- 4 forms:
  - Jump
    (page of 64 bytes)
    e.g. BR <addr>



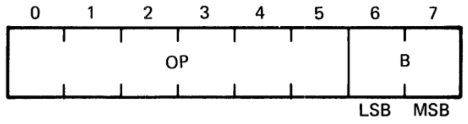
  - Immediate
    with nibble
    e.g. ALEC <val>



  - Immediate
    with X (2 bits)
    e.g. LDX <val>



  - No operand
    e.g. COMX, IA,
    TMA, XMA



- **Regular ?**
- **What is missing ?**

FUNCTION	MNEMONIC	STATUS EFFECTS		DESCRIPTION
		C	N	
Register to	TAY			Transfer accumulator to Y register.
Register	TYA			Transfer Y register to accumulator.
	CLA			Clear accumulator.
Transfer	TAM			Transfer accumulator to memory.
Register to	TAMIY			Transfer accumulator to memory and increment Y register.
Memory	TAMZA			Transfer accumulator to memory and zero accumulator.
Memory to	TMY			Transfer memory to Y register.
Register	TMA			Transfer memory to accumulator.
	XMA			Exchange memory and accumulator.
Arithmetic	AMAAC	Y		Add memory to accumulator, results to accumulator. If carry, one to status.
	SAMAN	Y		Subtract accumulator from memory, results to accumulator. If no borrow, one to status.
	IMAC	Y		Increment memory and load into accumulator. If carry, one to status.
	DMAN	Y		Decrement memory and load into accumulator. If no borrow, one to status.
	IA			Increment accumulator, no status effect.
	IYC	Y		Increment Y register. If carry, one to status.
	DAN	Y		Decrement accumulator. If no borrow, one to status.
	DYN	Y		Decrement Y register. If no borrow, one to status.
	A8AAC	Y		Add 8 to accumulator, results to accumulator. If carry, one to status.
	A10AAC	Y		Add 10 to accumulator, results to accumulator. If carry, one to status.
	A6AAC	Y		Add 6 to accumulator, results to accumulator. If carry, one to status.
	CPAIZ	Y		Complement accumulator and increment. If then zero, one to status.
Arithmetic	ALEM	Y		If accumulator less than or equal to memory, one to status.
Compare	ALEC	Y		If accumulator less than or equal to a constant, one to status
Logical	MNEZ		Y	If memory not equal to zero, one to status.
Compare	YNEA		Y	If Y register not equal to accumulator, one to status.
	YNEC		Y	If Y register not equal to a constant, one to status
Bits in	SBIT			Set memory bit.
Memory	RBIT			Reset memory bit.
	TBIT1		Y	Test memory bit. If equal to one, one to status.
Constants	TCY			Transfer constant to Y register.
	TCMIY			Transfer constant to memory and increment Y.
Input	KNEZ		Y	If K inputs not equal to zero, one to status.
	TKA			Transfer K inputs to accumulator.
Output	SETR			Set R output addressed by Y.
	RSTR			Reset R output addressed by Y.
	TDO			Transfer data from accumulator and status latch to O outputs.
	CLO			Clear O-output register.
RAM 'X'	LDX			Load 'X' with a constant.
Addressing	COMX			Complement 'X'.
ROM	BR			Branch on status = one.
Addressing	CALL			Call subroutine on status = one.
	RETN			Return from subroutine.
	LDP			Load page buffer with constant.

Sample instruction description

4-4.1 ADD MEMORY TO ACCUMULATOR, RESULTS TO ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	1	0	0	1	0	1

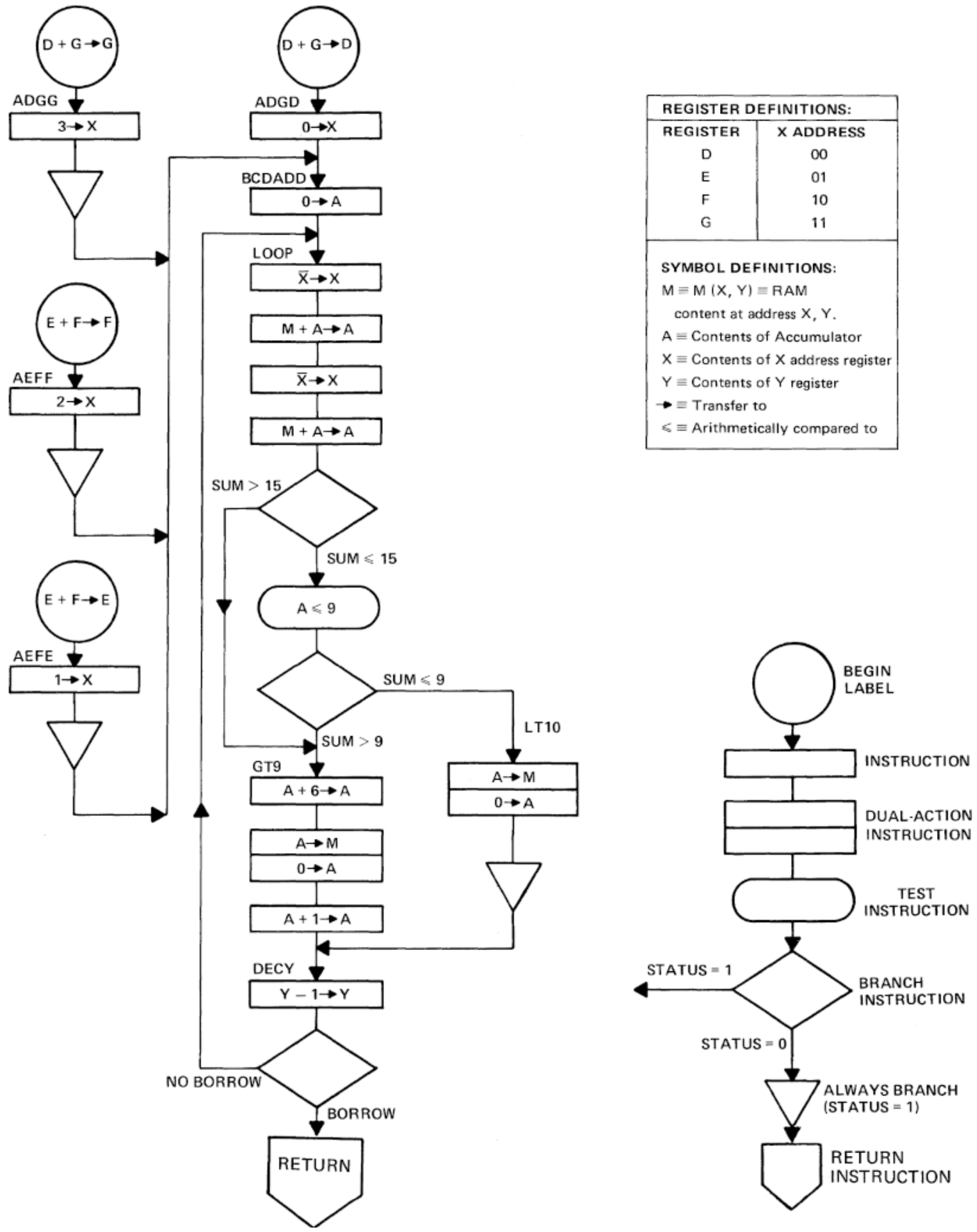
MNEMONIC:	AMAAC
STATUS:	Carry into status
FORMAT:	IV
ACTION:	$M(X,Y) + A \rightarrow A$ $1 \rightarrow S$ if sum > 15 $0 \rightarrow S$ if sum $\leqslant 15$
DESCRIPTION:	The contents of the memory location addressed by the X and Y registers are added to the contents of the accumulator. The result is stored into the accumulator. The resulting carry information is transferred to status. A sum that is greater than 15 results in a carry and a ONE to status. Memory contents are unaltered.
MICROINSTRUCTIONS:	MTP, ATN, C8, AUTA

Example BCD Addition

- Why BCD ? conversions, rounding, power of 10,…
- Representations: 1 nibbles = 1 digit but some unused (invalid valued)
  - Valid: 0->9
  - Invalid: A->F (1010, 1011, 1100, 1111)
- Rules for addition

- **Step 1** – Perform addition of two BCD numbers by following the rules of binary addition.
- **Step 2** – If the result or sum is a 4-bit binary number which is less than or equal to 9, then the sum is a valid BCD number.
- **Step 3** – If the sum is a 4-bit number that is greater than 9 or if a carry is generated, then it is an invalid sum.
- **Step 4** – To obtain the corrected result/sum, add 6 (0110) to the 4-bit invalid sum. If a carry is generated when 6 is added, then propagate and add this carry to the next 4-bit group. This step is done to skip the six illegal BCD codes (i.e. 1010, 1011, 1100, 1101, 1110, and 1111).

Example of BCD Addition



MULTIPLE ENTRY POINTS FOR SUBROUTINES	ADGG	LDX	3	3 → X; Set up for D + G → G.
		BR	BCDADD	Branch to BCD add.
	AEFF	LDX	2	2 → X; Set up for E + F → F.
		BR	BCDADD	Branch to BCD add.
	AEFE	LDX	1	1 → X; Set up for E + F → E.
		BR	BCDADD	Branch to BCD add.
	ADGD	LDX	0	0 → X; Add D + G → D.
BASE SUBROUTINE CONTAINS LOOPING AND BCD CORRECTION	BCDADD	CLA		Clear accumulator (A).
	LOOP	COMX		$\overline{X}$ → X.
		AMAAC		M(X,Y) + A → A; A contains possible carry if in loop.
		COMX		$\overline{X}$ → X.
		AMAAC		Add digits: M(X, Y) + [M($\overline{X}$,Y) + Carry] → A.
		BR	GT9	Branch if sum >15.
		ALEC	9	If A ⩽ 9, one to status.
		BR	LT10	Branch if sum < 10.
	GT9	A6AAC		Sum > 9, A + 6 → A; BCD Correction.
		TAMZA		Transfer corrected sum to memory, 0 → A.
		IA		1 → A; to propagate carry
	DECY	DYN		Y − 1 → Y; index next digit.
		BR	LOOP	If no borrow, continue.
		RETN		If borrow, return to instruction after call.
	LT10	TAMZA		Sum < 9, A → M(X,Y); 0 → A; No carry propagated.
		BR	DECY	

Example of BCD Addition

	LABEL	OPCODE	OPERAND	COMMENT
MAIN PROGRAM PRESETS Y, AND CALL SUBROUTINES		TCY	8	Transfer 8 → Y
		CALL	ADGD	Add: D + G → D
		TCY	15	Transfer 15 → Y
		CALL	AEFE	Add: E + F → E

FILE ADDRESS	REGISTER	Y-REGISTER ADDRESS															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
X = 00	D	OV 0	MSD 9	8	7	6	5	4	3	LSD 2							
X = 01	E	OV 0	MSD 1	2	3	4	5	6	7	8	9	0	1	2	3	4	LSD 5
X = 10	F	OV 0	MSD 5	4	3	2	1	0	9	8	7	6	5	4	3	2	LSD 1
X = 11	G	OV 0	MSD 8	7	6	5	4	3	2	LSD 1							

FILE ADDRESS	REGISTER	Y-REGISTER ADDRESS															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
X = 00	D	OV 1	MSD 8	6	4	1	9	7	5	LSD 3							
X = 01	E	OV 0	MSD 6	6	6	6	6	7	7	7	6	6	6	6	6	6	LSD 6
X = 10	F	OV 0	MSD 5	4	3	2	1	0	9	8	7	6	5	4	3	2	LSD 1
X = 11	G	OV 0	MSD 8	7	6	5	4	3	2	LSD 1							

Y=9:

9+7= 16 = 0x10

Corrected to 0+6=6 and A=1

Y=8:

8+8+1 is performed same way

etc

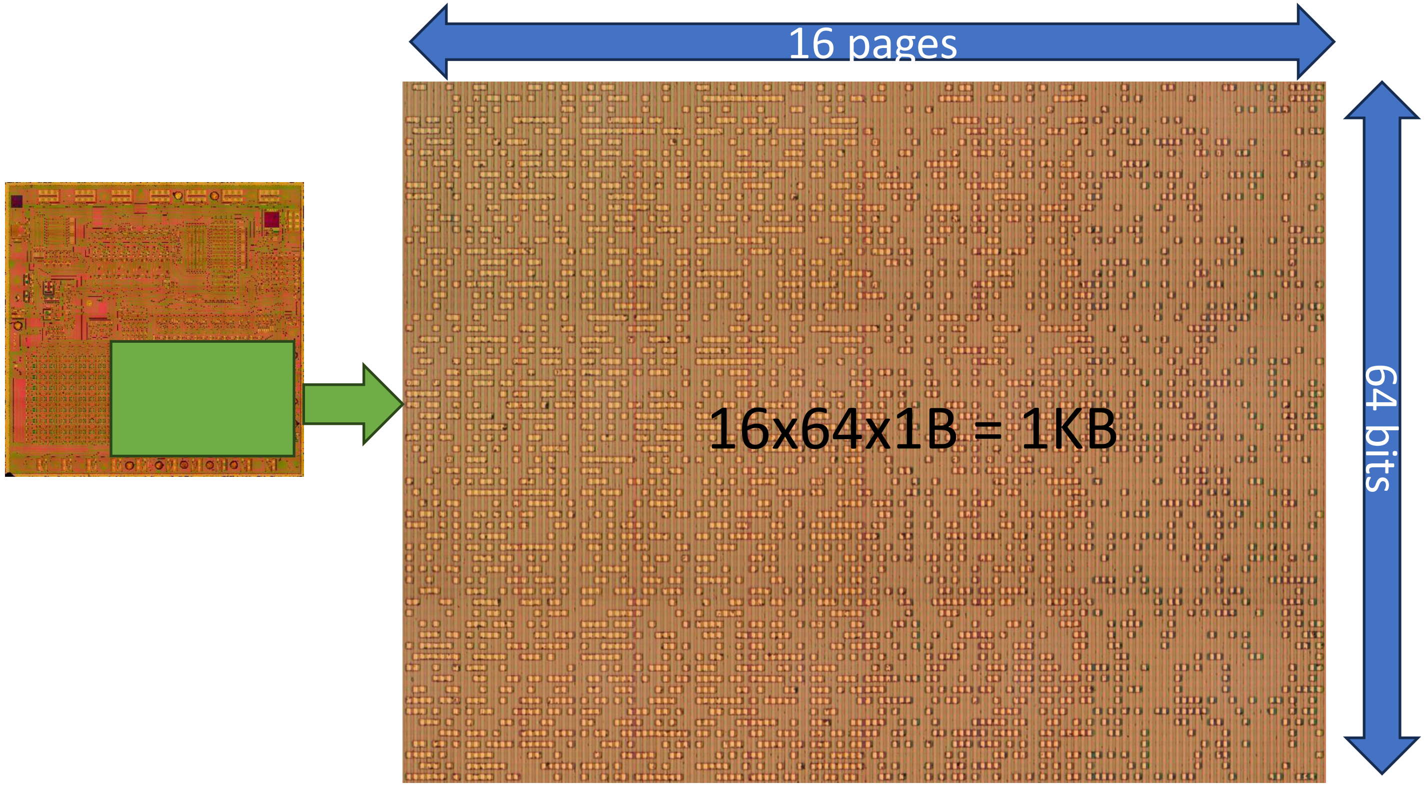
Reverse Engineering Little Professor (or other) ?

STEP #1 - ROM dump – answer to question ?

➔ Using TEST MODE (from patent) – no sure any success

➔ from die decaps and shots !

- thanks Sean Riddle - https://seanriddle.com/decap.html)
- requires reordering (patent helpful, can vary)
- See also: https://github.com/veniamin-ilmer/decoding_rom




```
mapping = [3, 4, 11, 12, 19, 20, 27, 28, 35, 36, 43, 44, 51, 52, 59, 60,
        0, 7, 8, 15, 16, 23, 24, 31, 32, 39, 40, 47, 48, 55, 56, 63,
        2, 5, 10, 13, 18, 21, 26, 29, 34, 37, 42, 45, 50, 53, 58, 61,
        1, 6, 9, 14, 17, 22, 25, 30, 33, 38, 41, 46, 49, 54, 57, 62]

# algo adapted from web but does not seem to match
def convert(tab):
    rx = 0
    res = [0] * (int)(len(tab)/8)
    for page in range(0,16):
        for pc in range(0,64):
            for bit in range(7,-1,-1):
                if page<8 :
                   rbi = pc*128+bit*16+page
                else:
                    rbi = pc*128+bit*16+23-page
                res[rx]=res[rx]*2+tab[rbi]
            rx = rx+1
    return res

def reorder(tab):
    res = [0] * len(tab)
    for i in range(0,len(tab)):
        r = i % 64
        d = i // 64
        res[i] = tab[d*64+mapping[r]]
        if (i<4): print(str(r)+" "+str(d)+" "+str(r)+" "+str(mapping[r])+" "+hex(res[i]))
    return res
```

['0x4f', '0x3c', '0x1e', '0x20', '0x32', '0xe4', '0x9b'…

Reverse Engineering ➔ decompiling

Can use assembler/disassembler - e.g. naken (GPL v3) - https://www.mikekohn.net/micro/naken_asm.php

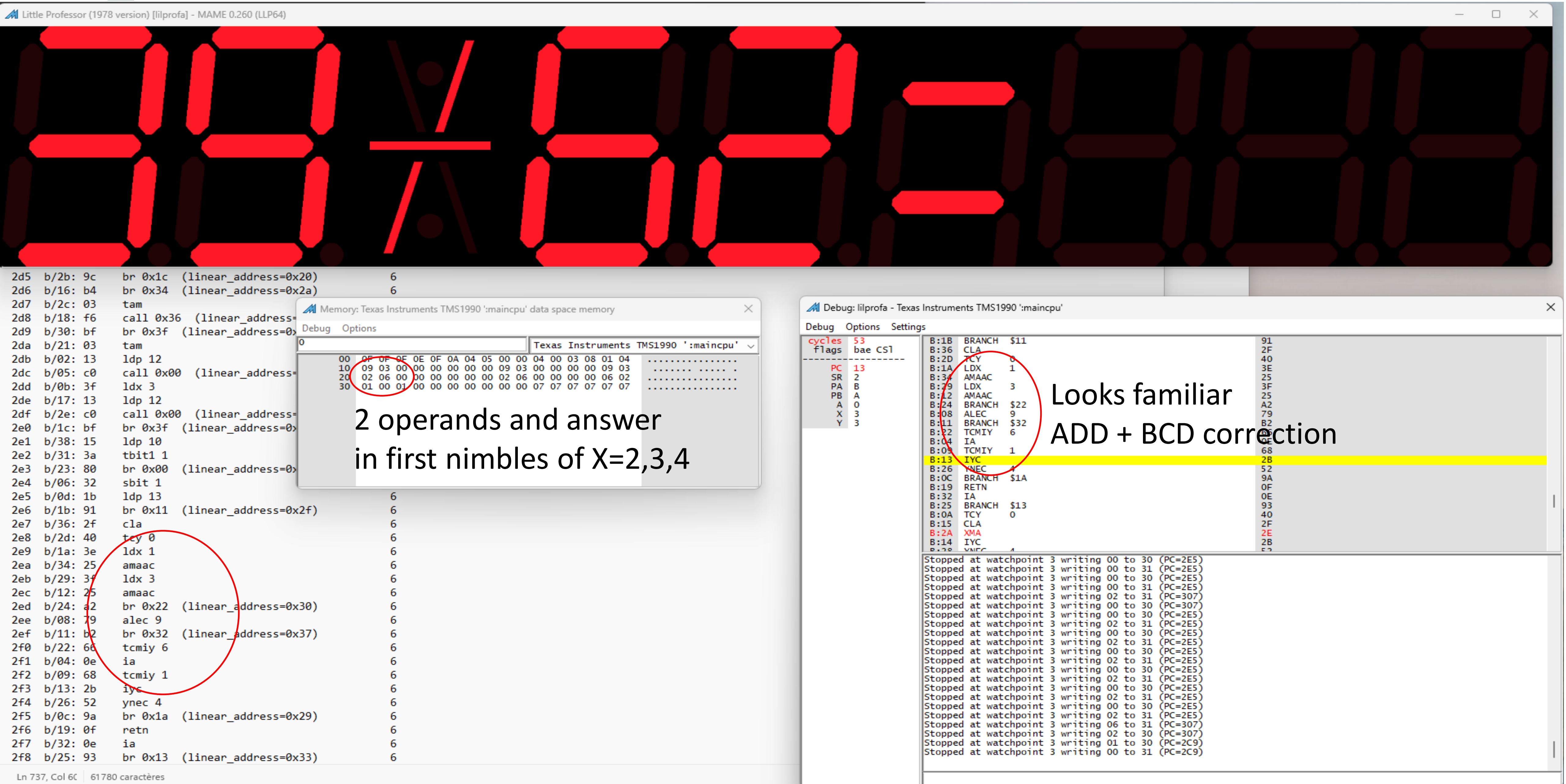
```
> naken_utils -disasm -tms1000 -bin tmc1993nl
```

```
naken_util - by Michael Kohn - Joe Davisson
    Web: http://www.mikekohn.net/
  Email: mike@mikekohn.net

Loaded bin tmc1993nl from 0x0000 to 0x03ff
Type help for a list of commands.

Linr Addr Opcode Instruction                             Cycles
---- ---- ------ ----------------------------------      ------
000  0/00: 4f    tcy 15                                  6
001  0/01: 3c    ldx 0                                   6
002  0/03: 20    tamiy                                   6
003  0/07: 43    tcy 12                                  6
004  0/0f: 32    sbit 1                                  6
005  0/1f: 4d    tcy 11                                  6
006  0/3f: 20    tamiy                                   6
007  0/3e: 4f    tcy 15                                  6
008  0/3d: 18    ldp 1                                   6
009  0/3b: 2a    dman                                    6
00a  0/37: 9e    br 0x1e   (linear_address=0x0c)         6
00b  0/2f: 4d    tcy 11                                  6
00c  0/1e: 9b    br 0x1b   (linear_address=0x26)         6
```

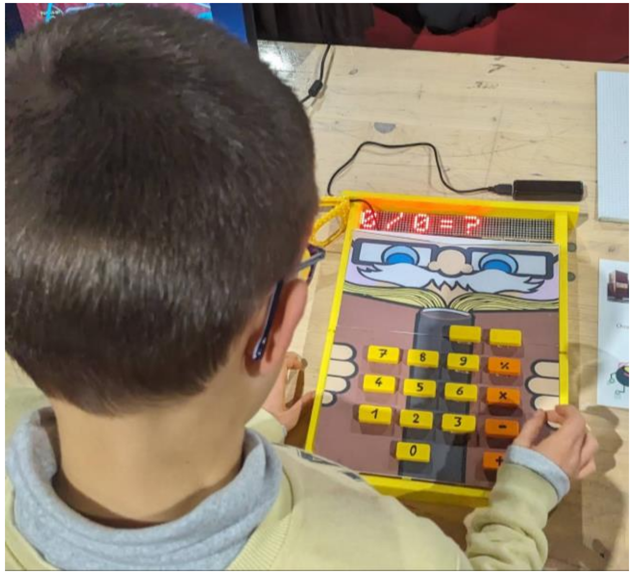
debug mode



Little Professor (1978 version) [lilprofa] - MAME 0.260 (LLP64)
2d5 b/2b: 9c br 0x1c (linear_address=0x20) 6
2d6 b/16: b4 br 0x34 (linear_address=0x2a) 6
2d7 b/2c: 03 tam
2d8 b/18: f6 call 0x36 (linear_address=
2d9 b/30: bf br 0x3f (linear_address=0x
2da b/21: 03 tam
2db b/02: 13 ldp 12
2dc b/05: c0 call 0x00 (linear_address=
2dd b/0b: 3f ldx 3
2de b/17: 13 ldp 12
2df b/2e: c0 call 0x00 (linear_address=
2e0 b/1c: bf br 0x3f (linear_address=0x
2e1 b/38: 15 ldp 10
2e2 b/31: 3a tbit1 1
2e3 b/23: 80 br 0x00 (linear_address=0x
2e4 b/06: 32 sbit 1
2e5 b/0d: 1b ldp 13 6
2e6 b/1b: 91 br 0x11 (linear_address=0x2f) 6
2e7 b/36: 2f cla 6
2e8 b/2d: 40 tcy 0 6
2e9 b/1a: 3e ldx 1 6
2ea b/34: 25 amaac 6
2eb b/29: 3f ldx 3 6
2ec b/12: 25 amaac 6
2ed b/24: a2 br 0x22 (linear_address=0x30) 6
2ee b/08: 79 alec 9 6
2ef b/11: b2 br 0x32 (linear_address=0x37) 6
2f0 b/22: 66 tcmiy 6 6
2f1 b/04: 0e ia 6
2f2 b/09: 68 tcmiy 1 6
2f3 b/13: 2b iyc 6
2f4 b/26: 52 ynec 4 6
2f5 b/0c: 9a br 0x1a (linear_address=0x29) 6
2f6 b/19: 0f retn 6
2f7 b/32: 0e ia 6
2f8 b/25: 93 br 0x13 (linear_address=0x33) 6
Ln 737, Col 60 61780 caractères
Memory: Texas Instruments TMS1990 ':maincpu' data space memory
Debug Options
Texas Instruments TMS1990 ':maincpu'
00 0F 0F 0F 0E 0F 0A 04 05 00 00 04 00 03 08 01 04
10 09 03 00 00 00 00 00 00 09 03 00 00 00 09 03
20 02 06 00 00 00 00 00 02 06 00 00 00 00 06 02
30 01 00 01 00 00 00 00 00 00 07 07 07 07 07 07
2 operands and answer
in first nimbles of X=2,3,4
Debug: lilprofa - Texas Instruments TMS1990 ':maincpu'
Debug Options Settings
cycles 53
flags bae CSl
PC 13
SR 2
PA B
PB A
A 0
X 3
Y 3
B:1B BRANCH $11 91
B:36 CLA 2F
B:2D TCY 0 40
B:1A LDX 1 3E
B:34 AMAAC 25
B:29 LDX 3 3F
B:12 AMAAC 25
B:24 BRANCH $22 A2
B:08 ALEC 9 79
B:11 BRANCH $32 B2
B:22 TCMIY 6 66
B:04 IA 0E
B:09 TCMIY 1 68
B:13 IYC 2B
B:26 YNEC 4 52
B:0C BRANCH $1A 9A
B:19 RETN 0F
B:32 IA 0E
B:25 BRANCH $13 93
B:0A TCY 0 40
B:15 CLA 2F
B:2A XMA 2E
B:14 IYC 2B
Looks familiar
ADD + BCD correction
Stopped at watchpoint 3 writing 00 to 30 (PC=2E5)
Stopped at watchpoint 3 writing 00 to 31 (PC=2E5)
Stopped at watchpoint 3 writing 00 to 30 (PC=2E5)
Stopped at watchpoint 3 writing 00 to 31 (PC=2E5)
Stopped at watchpoint 3 writing 02 to 31 (PC=307)
Stopped at watchpoint 3 writing 00 to 30 (PC=2E5)
Stopped at watchpoint 3 writing 00 to 31 (PC=2E5)
Stopped at watchpoint 3 writing 00 to 30 (PC=2E5)
Stopped at watchpoint 3 writing 00 to 31 (PC=2E5)
Stopped at watchpoint 3 writing 00 to 30 (PC=2E5)
Stopped at watchpoint 3 writing 02 to 31 (PC=2E5)
Stopped at watchpoint 3 writing 00 to 30 (PC=2E5)
Stopped at watchpoint 3 writing 02 to 31 (PC=2E5)
Stopped at watchpoint 3 writing 00 to 30 (PC=2E5)
Stopped at watchpoint 3 writing 02 to 31 (PC=2E5)
Stopped at watchpoint 3 writing 00 to 30 (PC=2E5)
Stopped at watchpoint 3 writing 02 to 31 (PC=2E5)
Stopped at watchpoint 3 writing 06 to 31 (PC=307)
Stopped at watchpoint 3 writing 02 to 30 (PC=307)
Stopped at watchpoint 3 writing 01 to 30 (PC=2C9)
Stopped at watchpoint 3 writing 00 to 31 (PC=2C9)

More Fun - From Little Professor to Big Professor

- Rebuilding in A3 format
  ➔ goal improve museum interactivity for kids
- Build from scratch using present day techniques: 3D printing, Arduino, LED panels
- Respect spirit: aspect, global experience
- But not strict
  - Emulating, not running “original ROMS” (but could)
  - Could play alternative games (guesser, box,…) of “cousins”
  - Better rewarding, e.g. retro animation (pacman, invaders,…)
- Nice feedback from first exhibitions
  - Parents recognise it
  - Kids play with it + look inside

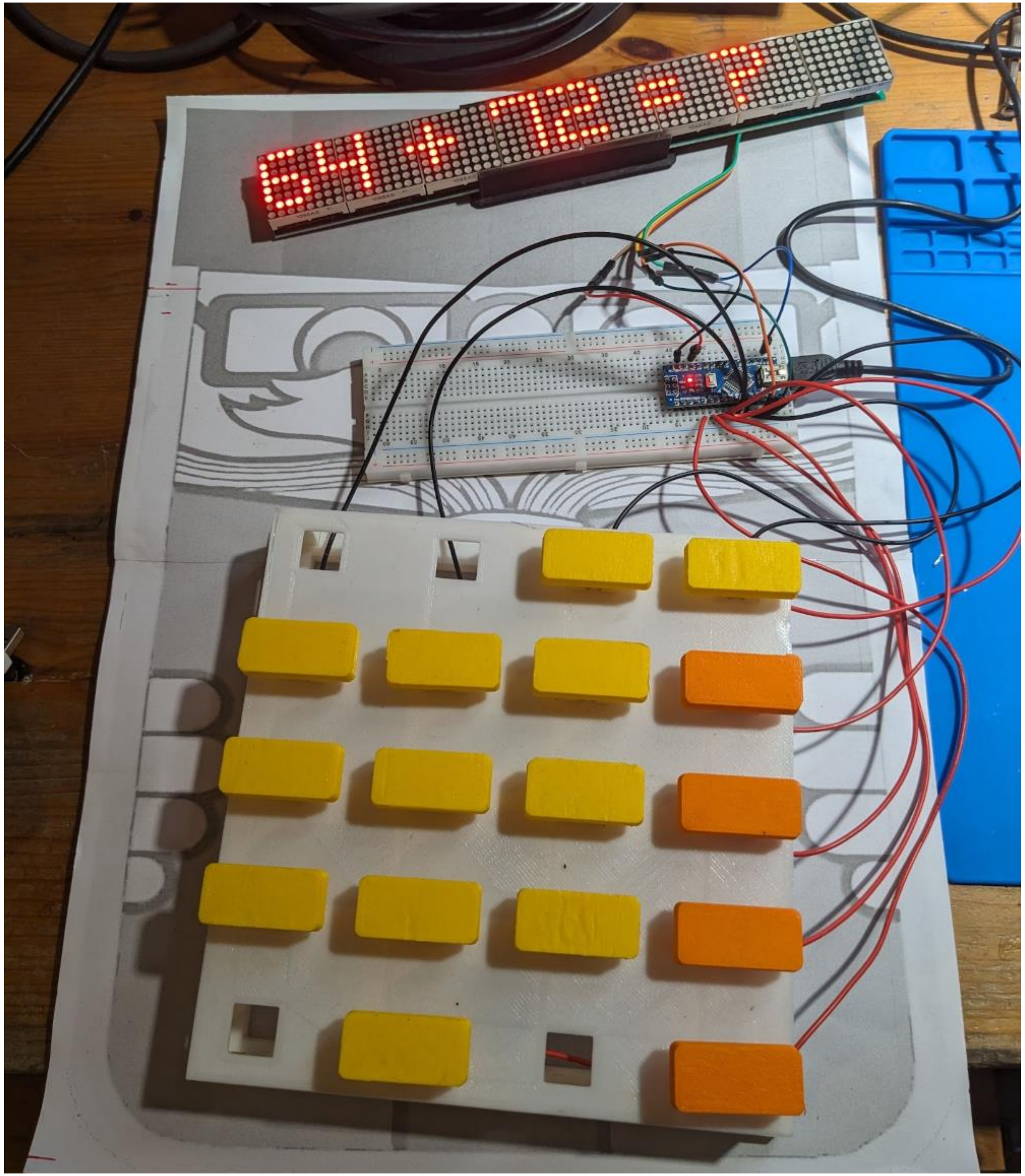


Quick Look Inside

- Keyboard matrix
  - cheap PC keys
  - scanned row/columns
  - similar to design
    (not intentionally)

- LED display:
  - serially addressed (SPI)
  - mimics red LED
  - more possibilities (for future)
    e.g. animations supported

- Open design available soon
  (thingiverse/instructables)



Conclusion

- Many discoveries starting from a donation
  - Rediscovery of an iconic game
  - Technical journey in the early days of microcontrollers/microprocessors
  - Preservation of rich history through emulation/rebuilding
  - Link between past and future

- On-going work: app version !
  - Using MIT app inventor ➔ also a way to learn coding to kids

Questions ?

Credits to:

- Sean Riddle for incredible die shots and so many ROM dumps
- Ken Shirriff for great reverse engineering and technology history @CHM
- My kids and Incubhacker (Namur) for “maker” support

Some references

- https://github.com/NAMIP-Computer-Museum/tms1000 (➔ curated resources)
- https://hackaday.com/2020/02/18/the-tms1000-the-first-commercially-available-microcontroller
- https://www.eejournal.com/article/a-history-of-early-microcontrollers-part-2-the-texas-instruments-tms1000

Visit us: www.nam-ip.be Twitter @ComputerMuseumB

Contact me: christophe.ponsard@gmail.com @cponsard @cponsard@ludosphere.fr