MONTANA
STATE UNIVERSITY

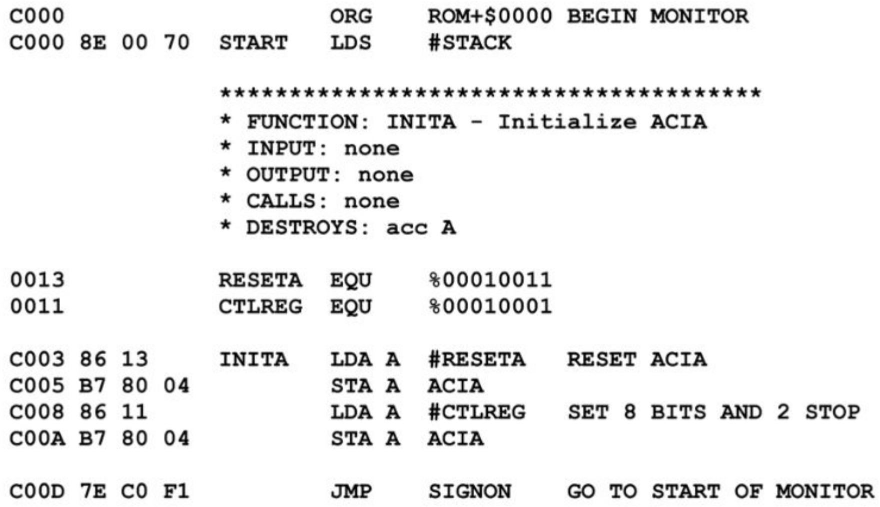
Introduction To Assembly

•••

Little Man Computer

Assembly Language

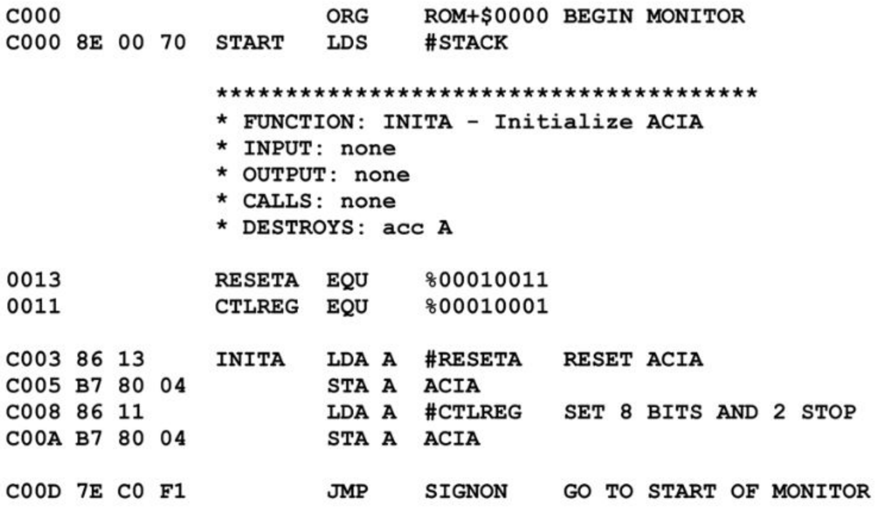
- What is assembly?
  - Assembly language is a low level programming language
  - Typically just above binary instructions
  - There is usually a 1 to 1 correspondence between assembly instructions and machine instructions



```
C000                   ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START   LDS    #STACK

               ***************************************
               * FUNCTION: INITA - Initialize ACIA
               * INPUT: none
               * OUTPUT: none
               * CALLS: none
               * DESTROYS: acc A

0013           RESETA  EQU    %00010011
0011           CTLREG  EQU    %00010001

C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
C005 B7 80 04          STA A  ACIA
C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04          STA A  ACIA

C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR
```

Assembly Language

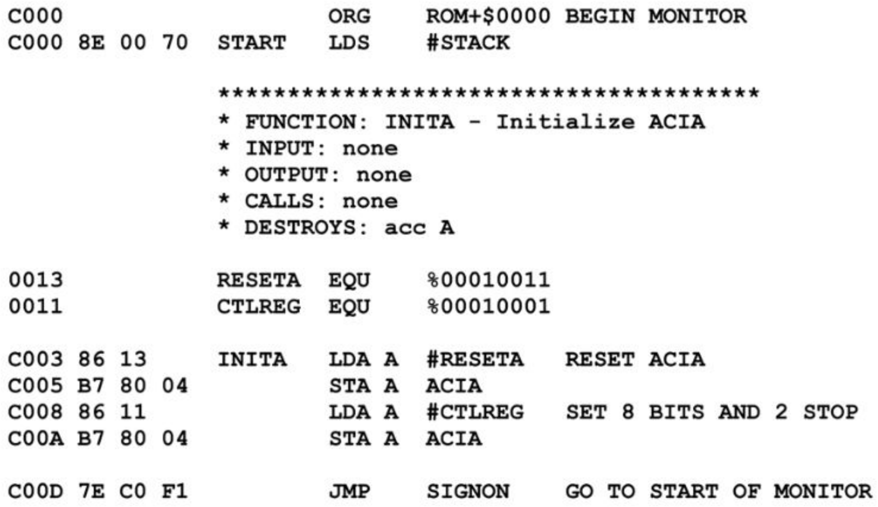
- Machine instructions == *machine dependent*
- Assembly language is specific to a particular architecture
- Recall the saying *“C is portable assembly”*
    - This is because the C compiler can target different architectures



```
C000                   ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START   LDS    #STACK

               ***************************************
               * FUNCTION: INITA - Initialize ACIA
               * INPUT: none
               * OUTPUT: none
               * CALLS: none
               * DESTROYS: acc A

0013           RESETA  EQU    %00010011
0011           CTLREG  EQU    %00010001

C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
C005 B7 80 04          STA A  ACIA
C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04          STA A  ACIA

C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR
```

Assembly Language

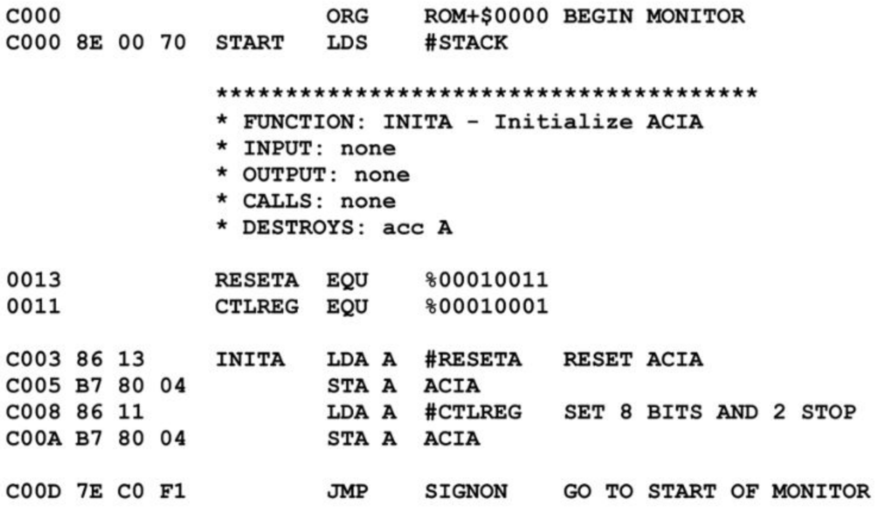
- Is C really portable assembly?
- Assembly typically does not have:
  - Function calls
  - Structs
  - Arrays
  - Loop constructs



```
C000                    ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70   START   LDS    #STACK

                ************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

0013            RESETA  EQU    %00010011
0011            CTLREG  EQU    %00010001

C003 86 13      INITA   LDA A  #RESETA   RESET ACIA
C005 B7 80 04           STA A  ACIA
C008 86 11              LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04           STA A  ACIA

C00D 7E C0 F1           JMP    SIGNON    GO TO START OF MONITOR
```

Assembly Language

- Is C really portable assembly?
- Assembly typically does have:
    - Jumps
    - Raw access to registers
    - A ton of work to get anything done

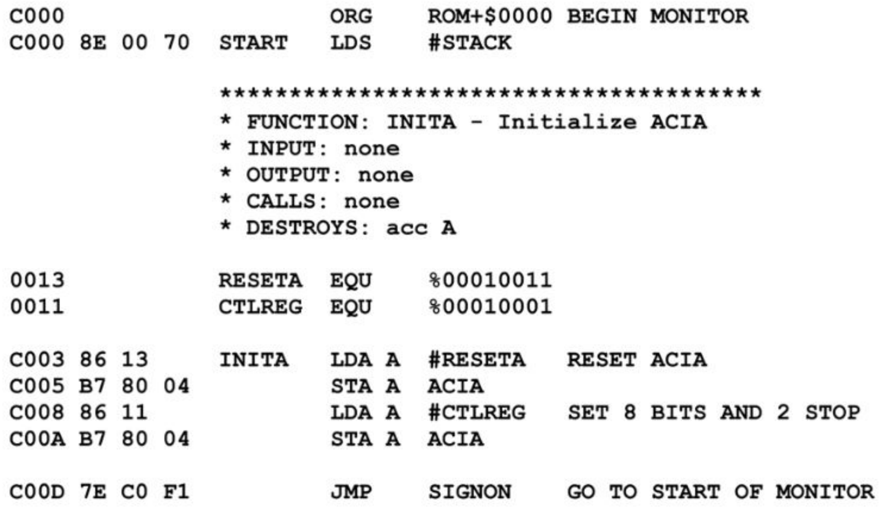


```
C000                   ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START   LDS    #STACK

               ***************************************
               * FUNCTION: INITA - Initialize ACIA
               * INPUT: none
               * OUTPUT: none
               * CALLS: none
               * DESTROYS: acc A

0013           RESETA  EQU    %00010011
0011           CTLREG  EQU    %00010001

C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
C005 B7 80 04          STA A  ACIA
C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04          STA A  ACIA

C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR
```

Assembly Language

- Is C really portable assembly?

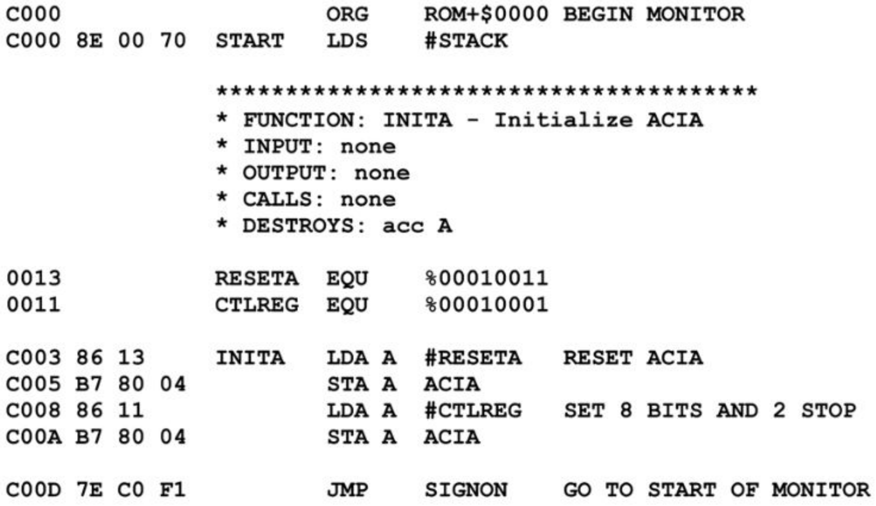
I would say that’s a good joke, but no, as low level as C is, it is much more than portable assembly



```
C000                    ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START    LDS    #STACK

               ***************************************
               * FUNCTION: INITA - Initialize ACIA
               * INPUT: none
               * OUTPUT: none
               * CALLS: none
               * DESTROYS: acc A

0013           RESETA   EQU    %00010011
0011           CTLREG   EQU    %00010001

C003 86 13     INITA    LDA A  #RESETA   RESET ACIA
C005 B7 80 04           STA A  ACIA
C008 86 11              LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04           STA A  ACIA

C00D 7E C0 F1           JMP    SIGNON    GO TO START OF MONITOR
```

Assembly Language

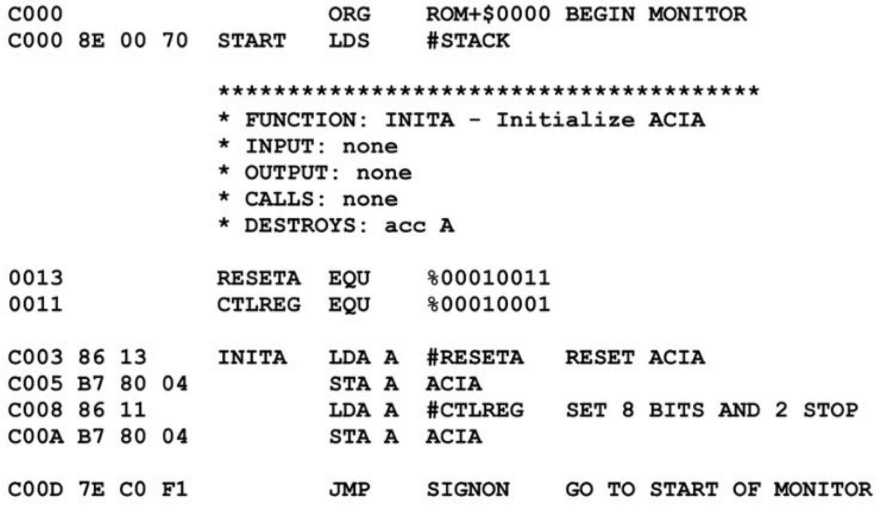
- Recall the assembler
- Takes assembly code and converts it into binary instructions
- Part of the gcc tool chain: C → assembly → object file → executable file



```
C000                    ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70   START   LDS    #STACK

                ****************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

0013            RESETA  EQU    %00010011
0011            CTLREG  EQU    %00010001

C003 86 13      INITA   LDA A  #RESETA   RESET ACIA
C005 B7 80 04           STA A  ACIA
C008 86 11              LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04           STA A  ACIA

C00D 7E C0 F1           JMP    SIGNON    GO TO START OF MONITOR
```

Assembly Syntax

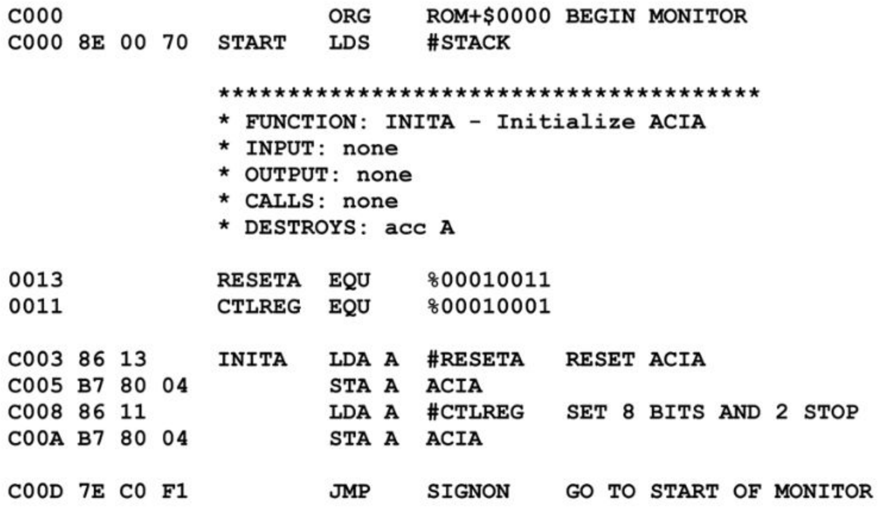
- Assembly code is typically a linear set of instructions, labels and directives
  - Assembly instructions correspond to single instructions on the CPU
  - Labels are used to refer to things by address
  - Directives can be comments, hints to the assembler, etc.



```
C000                   ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START   LDS    #STACK

               ***************************************
               * FUNCTION: INITA - Initialize ACIA
               * INPUT: none
               * OUTPUT: none
               * CALLS: none
               * DESTROYS: acc A

0013           RESETA  EQU    %00010011
0011           CTLREG  EQU    %00010001

C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
C005 B7 80 04          STA A  ACIA
C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04          STA A  ACIA

C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR
```

Understanding Assembly

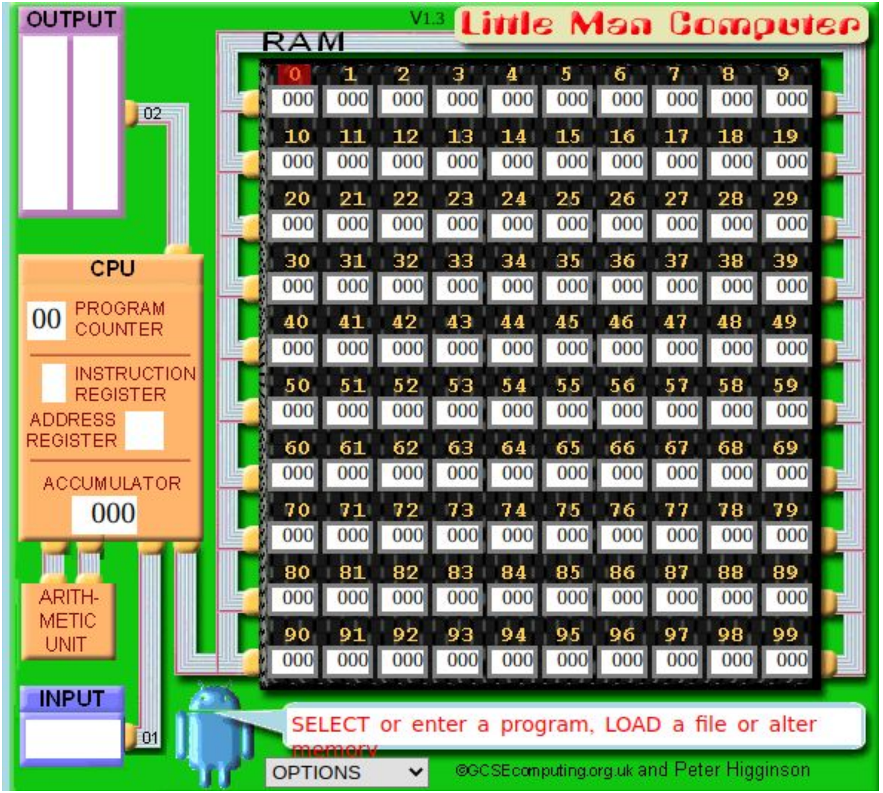
- To understand assembly, you have to understand the underlying hardware
- One of the difficult things about x86 is that the hardware is a little insane
- We are going to start with very simple hardware instead



```
C000                   ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START   LDS    #STACK

               ***************************************
               * FUNCTION: INITA - Initialize ACIA
               * INPUT: none
               * OUTPUT: none
               * CALLS: none
               * DESTROYS: acc A

0013           RESETA  EQU    %00010011
0011           CTLREG  EQU    %00010001

C003 86 13     INITA   LDA A  #RESETA   RESET ACIA
C005 B7 80 04          STA A  ACIA
C008 86 11             LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04          STA A  ACIA

C00D 7E C0 F1          JMP    SIGNON    GO TO START OF MONITOR
```

Little Man Computer

- The Little Man Computer (LMC)!
  - An extremely simple model of a computer
  - Proposed by Dr. Stuart Madnick in 1965
  - Models a simple von Neumann architecture machine
  - Has the basic operations of a modern computer, but much easier to understand



PROGRAM COUNTER: 00

INSTRUCTION REGISTER:

ADDRESS REGISTER:

ACCUMULATOR: 000

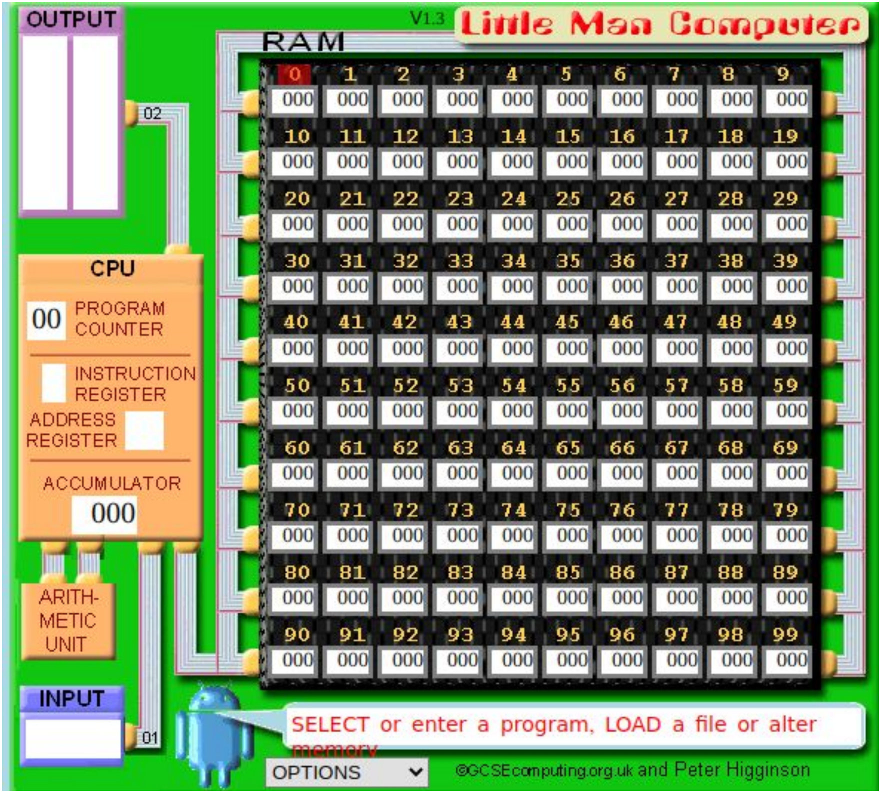
0	1	2	3	4	5	6	7	8	9
000	000	000	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

SELECT or enter a program, LOAD a file or alter memory

©GCSEcomputing.org.uk and Peter Higginson

Little Man Computer

- LMC simplifications
  - 100 memory slots
  - Addresses are in decimal
    - No binary or hex to confuse us
  - There is only one general register, called the *accumulator*
  - Values between -999 and 999 are supported



00 PROGRAM COUNTER

ACCUMULATOR 000

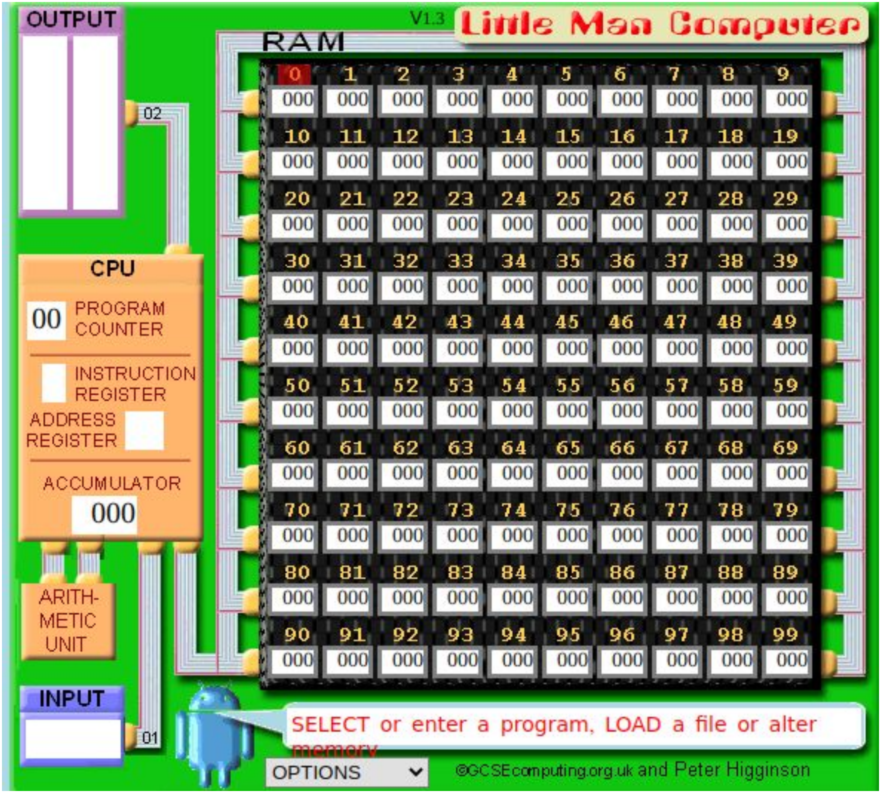
0	1	2	3	4	5	6	7	8	9
000	000	000	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

SELECT or enter a program, LOAD a file or alter memory

©GCSEcomputing.org.uk and Peter Higginson

Little Man Computer

- LMC Architecture
  - Program Counter - points to the current instruction (starts at 0)
  - Instruction Register - the current instruction being executed
  - Address Register - an address associated with the current instruction (if any)
  - Input/Output - Rudimentary I/O devices to read and print numbers



RAM

0	1	2	3	4	5	6	7	8	9
000	000	000	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

00 PROGRAM COUNTER

INSTRUCTION REGISTER

ADDRESS REGISTER

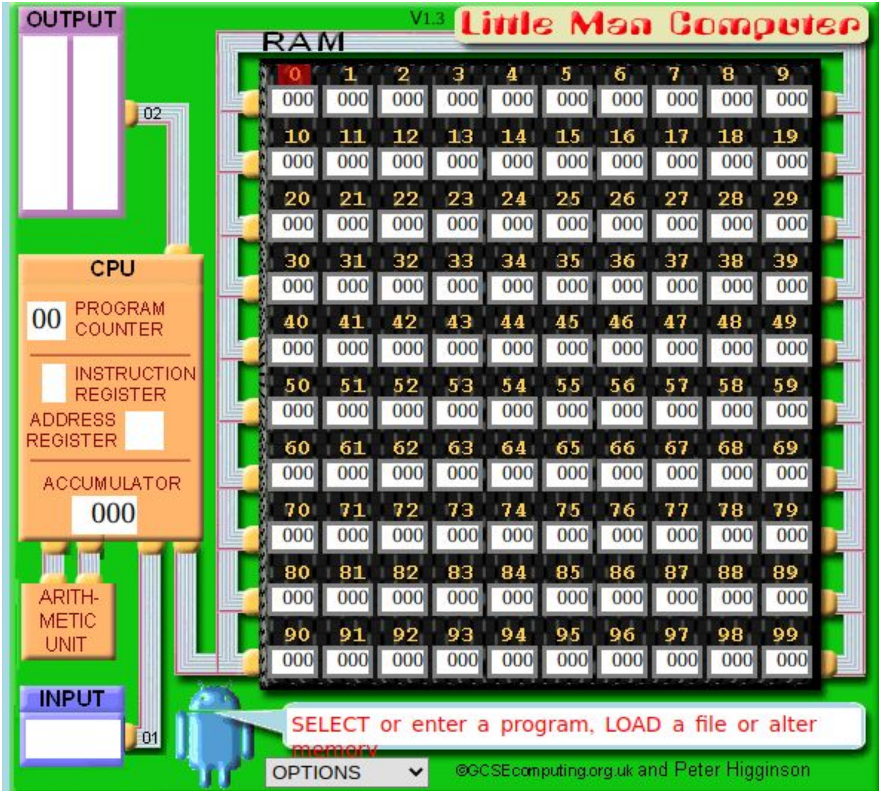
ACCUMULATOR 000

SELECT or enter a program, LOAD a file or alter memory

©GCSEcomputing.org.uk and Peter Higginson

Little Man Computer

- LMC Architecture
  - RAM - a continuous array of decimal values from position 0 to 100
  - Note that there is no distinction between instructions and data in memory



00 PROGRAM COUNTER

INSTRUCTION REGISTER

ADDRESS REGISTER

ACCUMULATOR 000

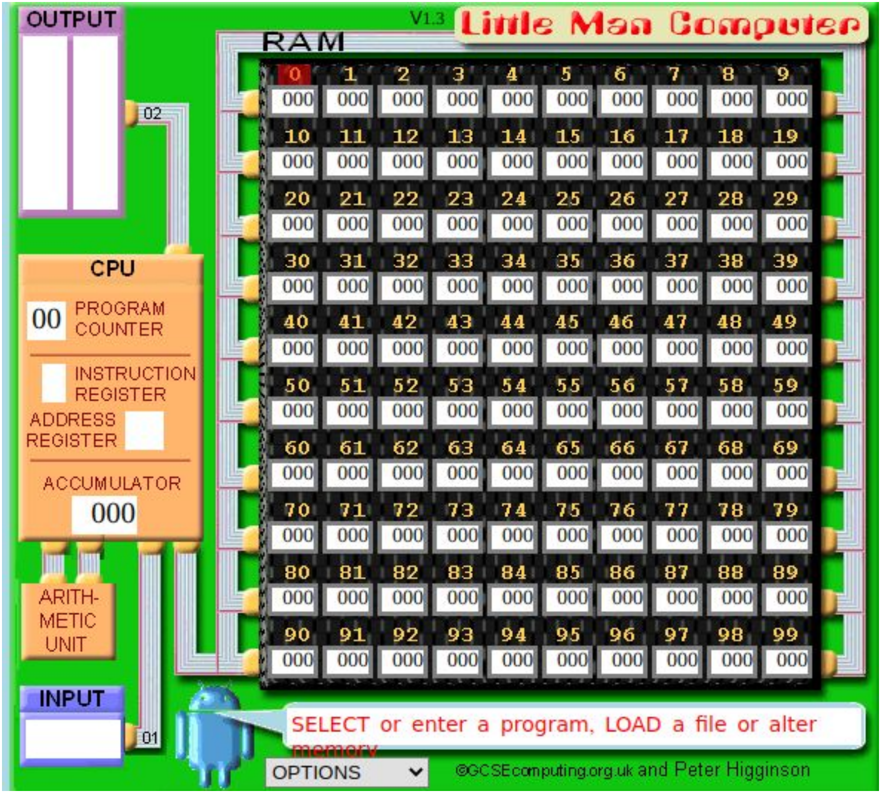
0	1	2	3	4	5	6	7	8	9
000	000	000	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

SELECT or enter a program, LOAD a file or alter memory

©GCSEcomputing.org.uk and Peter Higginson

Little Man Computer

- LMC Execution Cycle
  - Check the Program Counter
  - Fetch the instruction from that address
  - Increment the Program Counter
  - Decode the fetched instruction into the Instruction and Address registers
  - Fetch any data needed
  - Execute the instruction
  - Branch or store the result
  - Repeat!



00 PROGRAM COUNTER

INSTRUCTION REGISTER

ADDRESS REGISTER

ACCUMULATOR 000

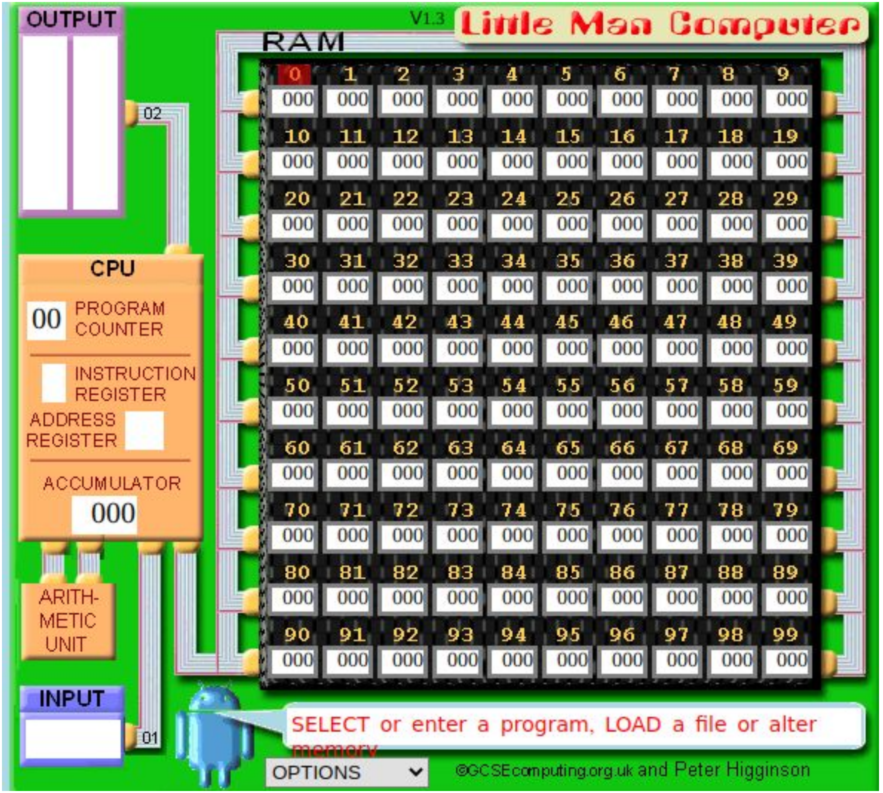
0	1	2	3	4	5	6	7	8	9
000	000	000	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

SELECT or enter a program, LOAD a file or alter memory

@GCSEcomputing.org.uk and Peter Higginson

Little Man Computer

- LMC Instruction Set
  - LMC Instructions are 3 decimals
  - First decimal indicates the instruction type
  - Next two decimals are optional arguments for that instruction



PROGRAM COUNTER 00

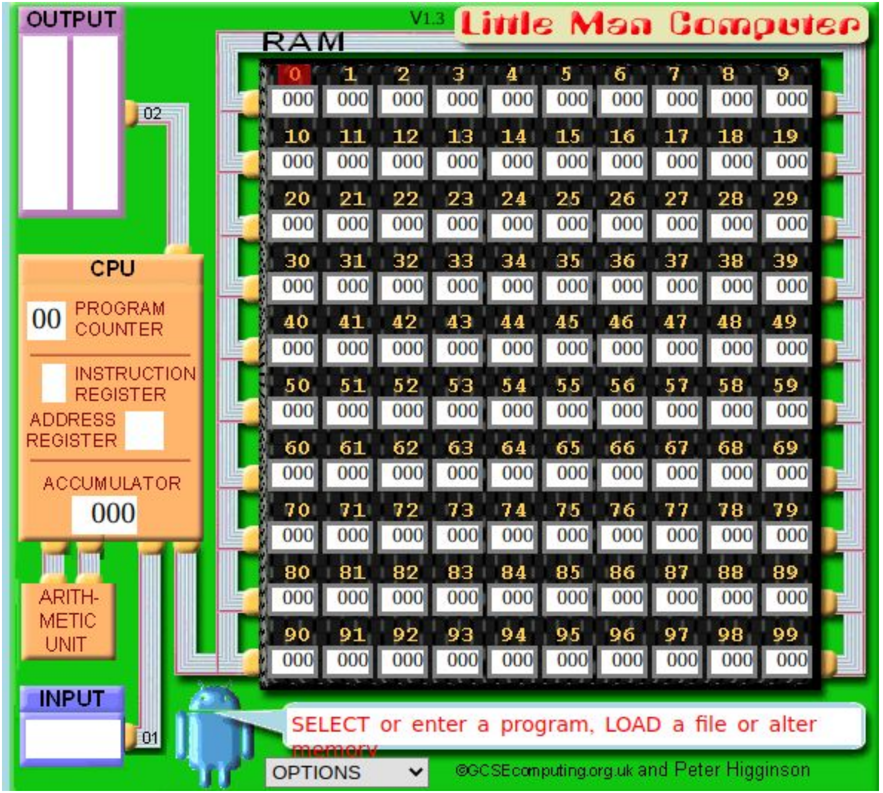
ACCUMULATOR 000

0	1	2	3	4	5	6	7	8	9
000	000	000	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

SELECT or enter a program, LOAD a file or alter memory

LMC - Math

- ADD
  - 1XX - adds the value stored in location XX to whatever value is currently in the Accumulator
- SUB
  - 2XX - subtracts the value stored in location XX from whatever value is in the Accumulator



PROGRAM COUNTER: 00

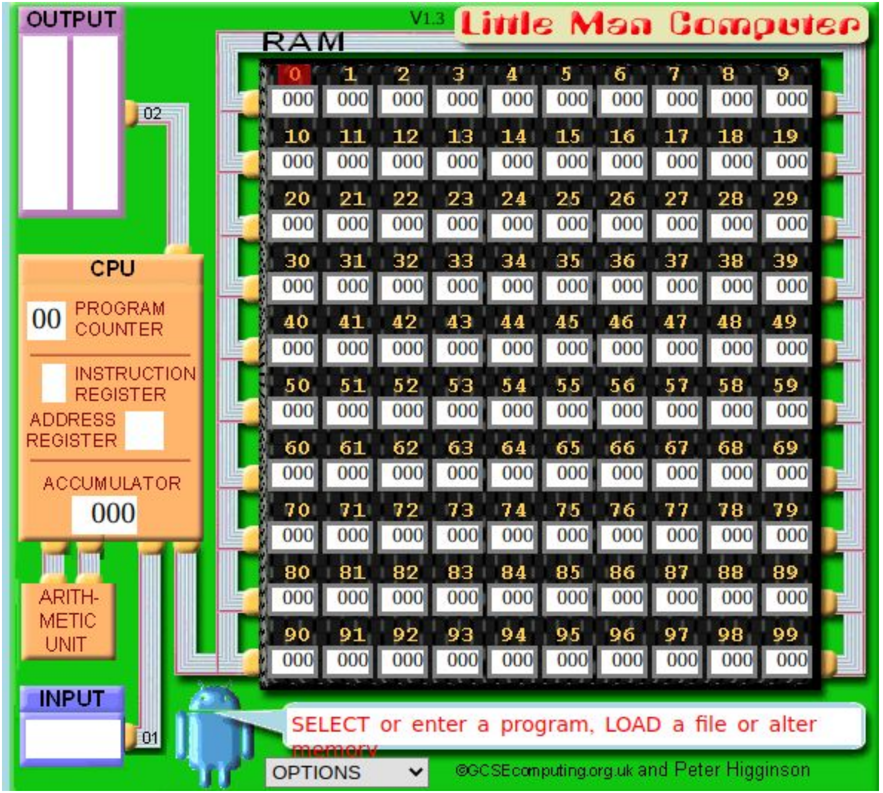
ACCUMULATOR: 000

0	1	2	3	4	5	6	7	8	9
000	000	000	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

SELECT or enter a program, LOAD a file or alter memory

LMC - Memory

- STA
  - 3XX - stores the value in the Accumulator into the memory location XX
- LDA
  - 5XX - loads the value in the memory location XX into the Accumulator



PROGRAM COUNTER 00

INSTRUCTION REGISTER

ADDRESS REGISTER

ACCUMULATOR 000

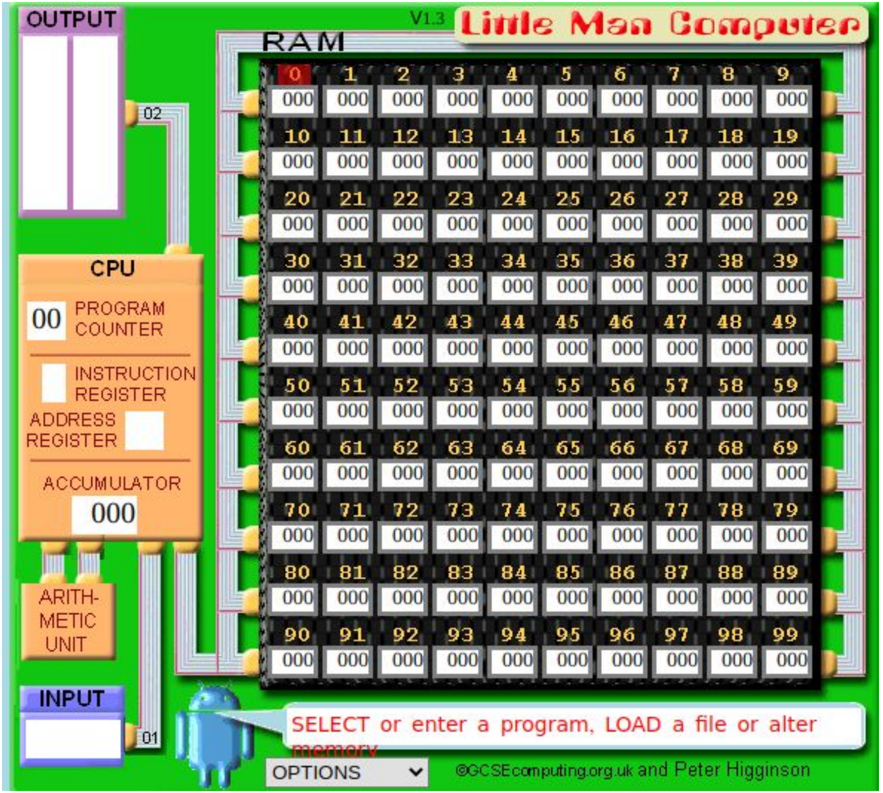
0	1	2	3	4	5	6	7	8	9
000	000	000	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

SELECT or enter a program, LOAD a file or alter memory

©GCSEcomputing.org.uk and Peter Higginson

LMC - Control Flow

- BRA
  - 6XX - unconditionally sets the Program Counter to the memory location XX
- BRZ
  - 7XX - branches to the location XX if the accumulator is zero
- BRP
  - 8xx - branches to the location XX if the accumulator is 0 or positive



PROGRAM COUNTER 00

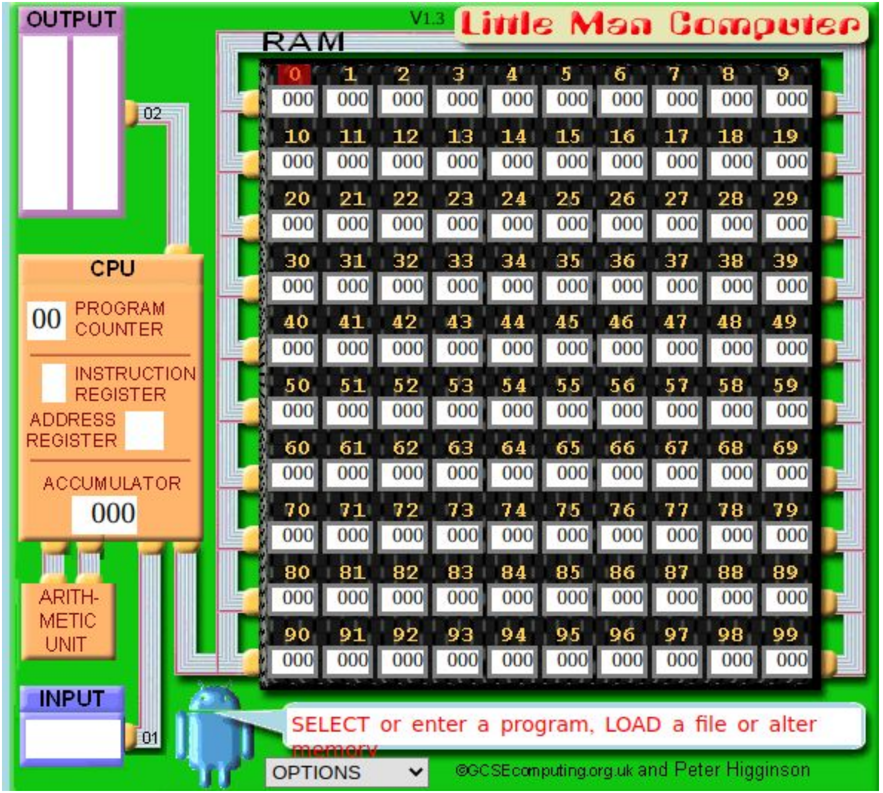
ACCUMULATOR 000

0	1	2	3	4	5	6	7	8	9
000	000	000	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

SELECT or enter a program, LOAD a file or alter memory

LMC - Input/Output

- INP
  - 901 - Ask the user for numeric input, to be stored in the Accumulator
- OUT
  - 902 - Write the current accumulator value to the output area
- HLT/COB
  - 000 - end the program, halt, take a coffee break



PROGRAM COUNTER 00

ACCUMULATOR 000

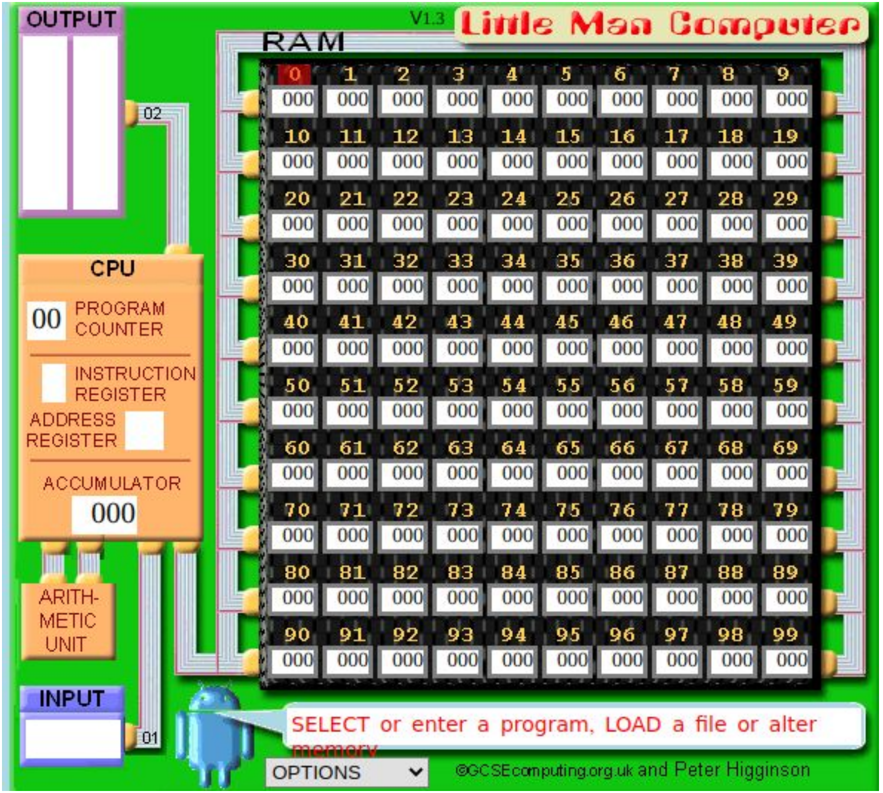
0	1	2	3	4	5	6	7	8	9
000	000	000	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

SELECT or enter a program, LOAD a file or alter memory

©GCSEcomputing.org.uk and Peter Higginson

LMC - Input/Output

- DAT <value>
  - Used to indicate that the value should be stored at the memory location corresponding to this instruction



PROGRAM COUNTER 00

ACCUMULATOR 000

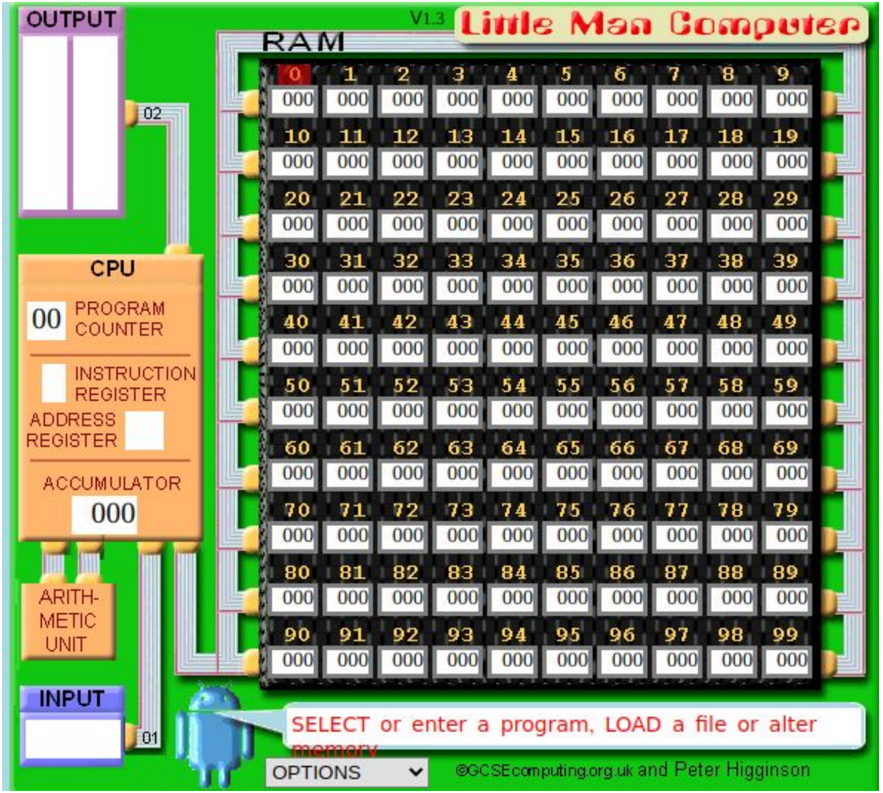
0	1	2	3	4	5	6	7	8	9
000	000	000	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

SELECT or enter a program, LOAD a file or alter memory

©GCSEcomputing.org.uk and Peter Higginson

LMC

- That’s it!
- A total of 10 instructions
- But we can still do some interesting things with them
- More complex architectures add more instructions and infrastructure (registers, etc.)
- But the fundamentals are the same



PROGRAM COUNTER 00

INSTRUCTION REGISTER

ADDRESS REGISTER

ACCUMULATOR 000

0	1	2	3	4	5	6	7	8	9
000	000	000	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

SELECT or enter a program, LOAD a file or alter memory

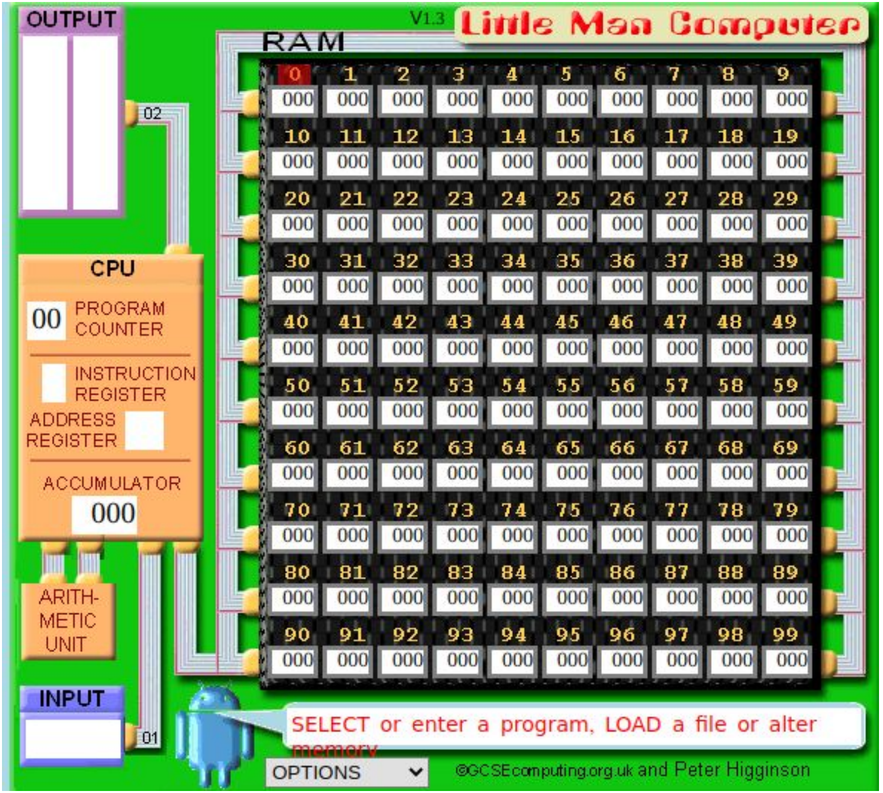
©GCSEcomputing.org.uk and Peter Higginson

LMC

- We will be using the excellent LMC simulator at

  https://peterhigginson.co.uk/LMC/

DEMO TIME!



00 PROGRAM COUNTER

INSTRUCTION REGISTER

ADDRESS REGISTER

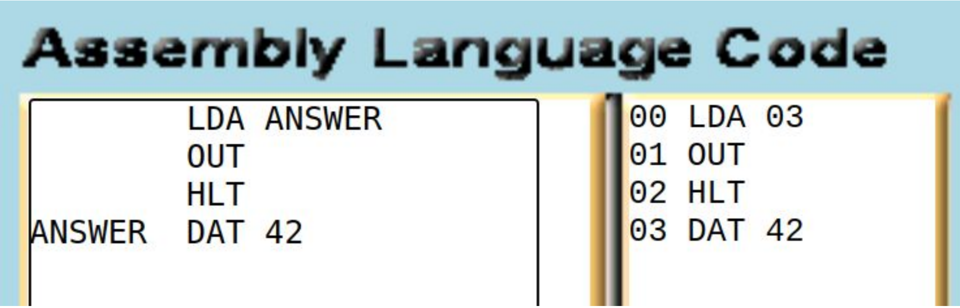
ACCUMULATOR 000

0	1	2	3	4	5	6	7	8	9
000	000	000	000	000	000	000	000	000	000
10	11	12	13	14	15	16	17	18	19
000	000	000	000	000	000	000	000	000	000
20	21	22	23	24	25	26	27	28	29
000	000	000	000	000	000	000	000	000	000
30	31	32	33	34	35	36	37	38	39
000	000	000	000	000	000	000	000	000	000
40	41	42	43	44	45	46	47	48	49
000	000	000	000	000	000	000	000	000	000
50	51	52	53	54	55	56	57	58	59
000	000	000	000	000	000	000	000	000	000
60	61	62	63	64	65	66	67	68	69
000	000	000	000	000	000	000	000	000	000
70	71	72	73	74	75	76	77	78	79
000	000	000	000	000	000	000	000	000	000
80	81	82	83	84	85	86	87	88	89
000	000	000	000	000	000	000	000	000	000
90	91	92	93	94	95	96	97	98	99
000	000	000	000	000	000	000	000	000	000

SELECT or enter a program, LOAD a file or alter memory

LMC - Labels

- A nice way to avoid having to hard code addresses into your assembly program
- Provide a *symbolic* way to refer to jump targets or data loads
- Here we are storing 42 at the 4th position and loading it via a label



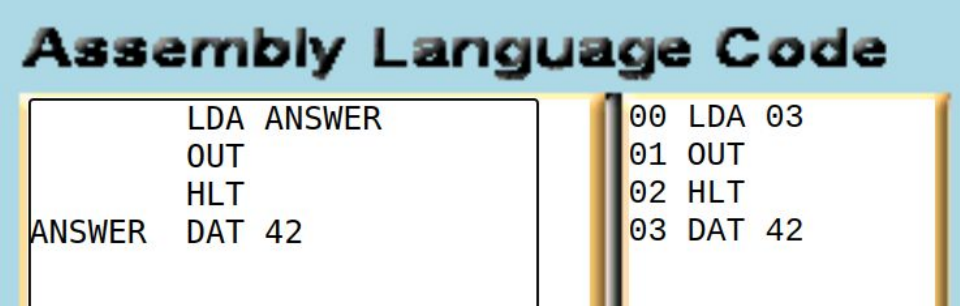
Assembly Language Code

```
        LDA ANSWER
        OUT
        HLT
ANSWER  DAT 42
```

```
00 LDA 03
01 OUT
02 HLT
03 DAT 42
```

LMC - Labels

- Labels also become important as we implement things like loops or conditional branches in our assembly program



Assembly Language Code

```
        LDA ANSWER
        OUT
        HLT
ANSWER  DAT 42
```

```
00 LDA 03
01 OUT
02 HLT
03 DAT 42
```

Assembly Review

- Assembly is very low level programming, typically just above machine code
- Assembly programs consist mainly of a linear sets of instructions, as well as data, assembly directives, etc.
- We are going to begin working with assembly using the LMC architecture
  - Simple and Fun!
  - Despite the simplicity, it shows us the core operations in any assembly language
- Next: implementing some stuff in LMC!

MONTANA
STATE UNIVERSITY