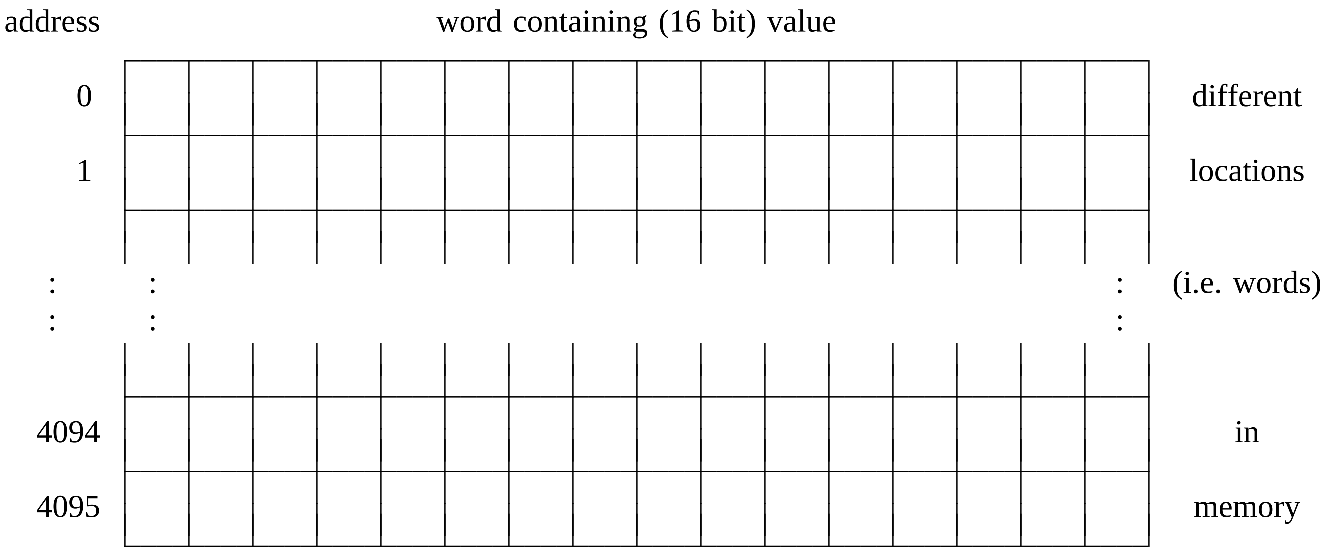
CS1011 lab exercise 4: The simulated MU0 computer

MU0 is a simple hypothetical computer. It has a simple internal architecture and instruction set which may be used as a basis for discussion of the operation of computers at a lower level than you will meet in most of the modules concerned with programming. Thus, we can use it to examine how some operations can be described by sequences of simple instructions, how these instructions are actually executed by the hardware and how this execution affects the current state of the computer.

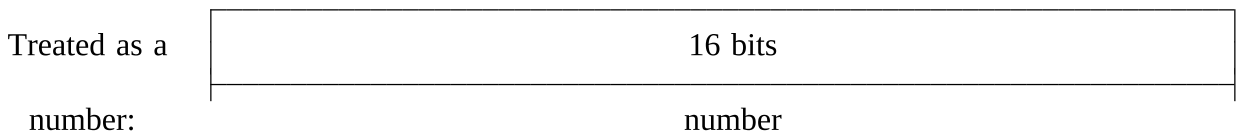
MU0 does not exist as a purchasable product, and its limited capabilities would suggest that no sales would be made if it did! However, we do have a software simulator for MU0 running, for the first time ever, on the department's NT PCs. This sheet tells you what this simulator does and how to use it.

MU0

The MU0 machine has 4K (i.e. 4096) words of memory. Each word can hold a value. To locate words and thus their values, each word has a different address, from 0 to 4095.



Each value consists of 16 binary digits, or bits, and can be used either as if it is a signed integer number, in the range -32768 to 32767, or as if it is a computer instruction. Each 16 bits holding a computer instruction is treated as if it consisted of two components, a 4 bit function (i.e. which operation to perform) and a 12 bit address (i.e. which location in memory to perform the operation on).



Treated as an	4 bits	12 bits
instruction:	function	address

An address component of 12 bits is just enough to hold 4096 different addresses, and the simplest way of treating these is as a number from 0 to 4095. Similarly, a function component of 4 bits could hold any of 16 different values. However, MU0 is so simple that it only uses 8 of these different values, which we will refer to as functions 0 to 7.

There are four functions used to make calculations using signed, whole numbers. Arithmetic is actually performed using another 16 bit location which is not in the memory, but is instead referred to by name, the Accumulator register (ACC).

Functions for calculations:

name	number	action
LDA	0	load the contents of the memory location to the ACC
STO	1	store the number in the ACC to the memory location
ADD	2	add the contents of the memory location to the ACC
SUB	3	subtract the contents of the memory location from the ACC

Thus we could add the contents of locations 1000 and 1001 and put the result into location 2000 by:

```
LDA 1000     load the contents of memory location 1000 into the ACC
ADD 1001     add the contents of memory location 1001 to the ACC
STO 2000     store the value in the ACC to memory location 2000
```

The four control functions are used to alter the sequence in which the instructions are obeyed, which is controlled by another named location, the Program Counter register (PC). This has 12 bits, and contains the location of the next value in memory which will be treated as an instruction and obeyed. Although this is normally just incremented as each instruction is executed, so that instructions are obeyed sequentially, the JMP function overwrites the value in PC by the address part of the instruction. Thus it can be used to skip over instructions, or to go back and repeat them. The two conditional jump functions, JGE and JNE, give us the ability to make simple decisions. These instructions may change the value in PC or not, depending on the value currently in the ACC. Finally, the STP instruction is used to stop a program once it has completed all it is supposed to do.

Control functions:

name	number	action
JMP	4	jump to the memory location (i.e. treat it as containing an instruction)
JGE	5	jump to the memory location if the ACC is ≥ 0
JNE	6	jump to the memory location if the ACC is $\neq 0$
STP	7	stop executing instructions

The simulator form of MU0 instructions

The simulator for MU0, currently available on the Windows NT PCs, permits you to input MU0 instructions in symbolic assembler form, to edit them, to execute them, to step through execution one instruction at a time and to view the instructions, ACC and PC. The format for instructions is one per line as follows:

label: spaces or tab instruction or number spaces or tab comment

The "label:" (and the following spaces or tab) need not be present, but if it is present it must be a letter followed by up to 5 letters and digits before the ":".

The instruction must be one of the 3 letter function names listed above, followed by an address (except that the "STP" function does not need one). An address is either a decimal number or the name of a label (which is taken to mean the address of the location where the corresponding instruction or number is stored). A single number may be given instead of an instruction. The instruction or number is converted into a 16 bit value and stored in the next memory location, starting at location 0.

The comment is optional, and consists of any other characters on the line.

Exercises

The purpose of this practical session is to give you some awareness of the nature of machine code instructions and how sequences of them are executed. This is a topic covered in more depth in the CS1031 module, but is useful background for anyone using a computer or studying applications. Exercises, or parts of exercises, surrounded by "[[" "]]" should be considered as optional extras for those students who are making quick progress.

Start the simulator:

Find a Windows NT PC, log in as normal. Find MU0 by double-clicking the MU0 subdirectory on the O: drive. Then select MU0 by double clicking on the MU0 icon, wait for the MU0 simulator to put up a window, with several sub-windows.

At present the Load File option on the file menu crashes MU0, so do not use this, it is not necessary for today’s exercises. Also when you finally exit MU0 you may get a ’stack overflow’ error which should also be ignored.

Exercise 4.1

Type in the following program text in the top, left-hand sub-window, entitled MU0 source:

```
        LDA     JACK
        STO     JILL
        STP
JACK:   1
JILL:   2
```

(Note that the : must follow immediately after JACK with no space. The : must be followed by a space or tab.) Select **ASSEMBLE** from the **RUN** menu, using the left mouse button, to check the program. If this is successful, the equivalent binary program appears in the top, right-hand sub-window e.g.:

    0000: 0003 LDA JACK

"0000:" represents the address
"0003" is a hexadecimal representation of the instruction
"LDA JACK" reminds you of the human-readable form of the instruction

If you get an Errors pop-up, or the right-hand window does not display sensible information, you may have mistyped the program. Click OK on the Errors pop-up and the errors window will be displayed. Each line in this window will be a error or warning message about what you have typed. Correct the errors and try **ASSEMBLE** again. If you can't see the problem, ask for help.

Assuming your program is correct, select **RUN RUN CONTINUOUS** to run the program. Note how the top right-hand sub-window changes, representing changes in MU0 as the program runs.

Select **SYMBOL TABLE** from the **VIEW** menu. This will display, in a separate sub-window, a symbol table which gives the address corresponding to each label. For example, in this program "JACK" means address 3. You might like to resize and move this window to an empty part of the screen.

Now select **RESET** from the **RUN** menu to reset the state of the MU0 the simulator to as it was before the program was run. Now execute the instructions one by one using **RUN RUN STEP** . It is very tedious to do this repeatedly and more convenient to click the shortcut button which shows a small green square beside a green right arrow. Note the changing values of ACC and PC at the top of the right-hand sub-window and the memory locations JACK and JILL (also in the top right-hand sub-window).

What memory location has changed as a result of running this program, and why?

Do you understand clearly the operation of each instruction and the relationship of it to the ACC and memory? If not, ask for advice and help.

Exercise 4.2

Edit your text to be as follows:

```
        LDA     JACK
        ADD     JILL
        ADD     TOM
        STO     TINA
        STP
JACK:   1
JILL:   2
TOM:    3
TINA:   0
```

ASSEMBLE and RUN this as for exercise 4.1. Do you now understand clearly the operation of the ADD instruction? If not, ask for advice and help.

[[Optional extra - only do this if you have plenty of time to spare:
MU0 is so simple that it does not have a multiply instruction. Edit the program to perform a suitable set of ADDs and STOres to multiply the contents of JACK by 10 and place the result in TINA.]]

Exercise 4.3

Edit your text to be as follows:

```
LOOP:   LDA   TOM
        ADD   JILL
        STO   TOM
        LDA   JACK
        SUB   TINA
        STO   JACK
        JNE   LOOP
        STP
JACK:   3
JILL:   4
TOM:    0
TINA:   1
```

ASSEMBLE and RUN this as with the previous exercises. Note particularly the changing values of ACC and PC as the JNE instruction is executed. What does the program do?

Now select **CPU window** from the **VIEW** menu. Move the "MU0 - CPU Visual" window so that it does not cover the original window. Try **STEP**ping through the program; the simulator will highlight the data movement around MU0. Each instruction corresponds to several substeps. Try to understand what is happening in each substep. At each substep inactive parts of the MU0 machine are green, the value used in the step is blue and the active parts of the machine are red. The meaning of these substeps will be made clear when we discuss the internal structure of MU0 in a later lecture. After the lecture, repeat this part of the exercise, verifying what you see against the explanation given in the lecture.

[[Optional extra - only do this if you have plenty of time to spare:
Will the program work with negative numbers? Edit the program so that it will work efficiently whether JACK and JILL contain positive or negative numbers.]]

[[Optional extra - only do this if you have plenty of time to spare:
Edit the program to perform a division by continuous subtraction, with positive or negative numbers.]]

Answers to MU0 lab exercises.

Exercise 4.2 - optional part (multiply by 10)

(note: anything after an instruction is a comment)

```
        LDA   JACK   * 1
        ADD   JACK   * 2
        ADD   JACK   * 3
        ADD   JACK   * 4
        ADD   JACK   * 5
        ADD   JACK   * 6
        ADD   JACK   * 7
        ADD   JACK   * 8
        ADD   JACK   * 9
        ADD   JACK   * 10
        STO   JILL
        STP
JACK:   1
JILL:   0
```

or you can shorten it by e.g.:

```
        LDA   JACK   * 1
        ADD   JACK   * 2
        STO   JILL
        ADD   JILL   * 4
        ADD   JILL   * 6
        ADD   JILL   * 8
        ADD   JILL   * 10
        STO   JILL
        STP
JACK:   1
JILL:   0
```

Exercise 4.3 - optional part 1 (negative numbers)

```
        LDA   JACK      if jack is positive
        JGE   LOOP      then ok else
        LDA   ANSWER      negate jack (by subtracting it from zero)
        SUB   JACK
        STO   JACK
        LDA   ANSWER      negate jill
        SUB   JILL
        STO   JILL
LOOP:   LDA   ANSWER    loop as before
        ADD   JILL
        STO   ANSWER
        LDA   JACK
```

```
            SUB     ONE
            STO     JACK
            JNE     LOOP
            STP
JACK:       3
JILL:       4
ANSWER:     0
ONE:        1
```

Exercise 4.3 - optional part 2 (division)

```
            LDA     JACK        if jack is negative then
            JGE     OK
            LDA     ANSWER        negate jack (by subtracting it from zero)
            SUB     JACK
            STO     JACK
            LDA     ANSWER        negate jill
            SUB     JILL
            STO     JILL
OK:         LDA     JILL        if jill is negative then
            JGE     LOOP
            LDA     ANSWER        negate jill
            SUB     JILL
            STO     JILL
            LDA     ANSWER        set answer to be negated (by negating one)
            SUB     ONE
            STO     ONE
LOOP:       LDA     JACK        subtract divisor from quotient
            SUB     JILL
            STO     JACK
            JGE     IFGE        if <0 then stop
            STP
IFGE:       LDA     ANSWER      else increment answer
            ADD     ONE
            STO     ANSWER
            JMP     LOOP        and repeat
JACK:       3
JILL:       4
ANSWER:     0
ONE:        1
```