# Assignment 3
# RTL Design

*Part 1 due Monday, February 28 2000*
*Part 2 due Monday, March 6 2000*

In this two-part assignment you will design and test a simple stack-based computer. Each part will be marked and counted as one assignment.

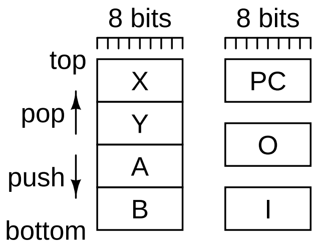
## Specifications

### Processor Architecture

The CPU has a 4-element LIFO (last-in first-out, or "push-down") stack, a program counter register (PC), and an output port register (O). There is also an input port (I), a reset input and clock input.

The stack's registers are labelled from top to bottom as: X, Y, A and B.

All registers, the input port and the program memory are 8 bits wide.



The processor has a 32-byte program memory whose contents are fixed (it is a ROM).

### Instruction Set

The CPU retrieves an instruction from the ROM location addressed by PC and executes it in one clock cycle.

Each instruction causes the stack contents to move up (pop), move down (push) or neither (none). Many instructions load X with a different value than would be given by the stack movement. One instruction (SWAP) results in no stack movement but loads Y with the contents of X.

The following table summarizes the instruction set. It gives the instruction mnemonics, the corresponding stack movement direction, and the registers loaded by each instruction are:

| Opcode | Mnemonic | Stack Movement | Instruction Effect |
|---|---|---|---|
| 0–127 | `PUSH` | down | X = instruction |
| 129+n | `POP` | up | - |
| 130+n | `INP` | down | X = I |
| 131+n | `OUTP` | up | O = X |
| 132+n | `DUP` | down | X = X |
| 133+n | `SWAP` | none | X = Y, Y = X |
| 134+n | `ADD` | up | X = Y+X |
| 135+n | `SUBT` | up | X = Y-X |
| 136+n | `JNZ` | up | if Y is non-zero PC = X |
| 137+n | `CALL` | none | PC = X, X = PC+1 |
| 138+n | `RET` | up | PC = X |

The new stack contents are computed using the stack contents *before* the stack is changed. Registers not shown above will take on the values resulting from the normal stack movement. Moving the stack up copies B into B.

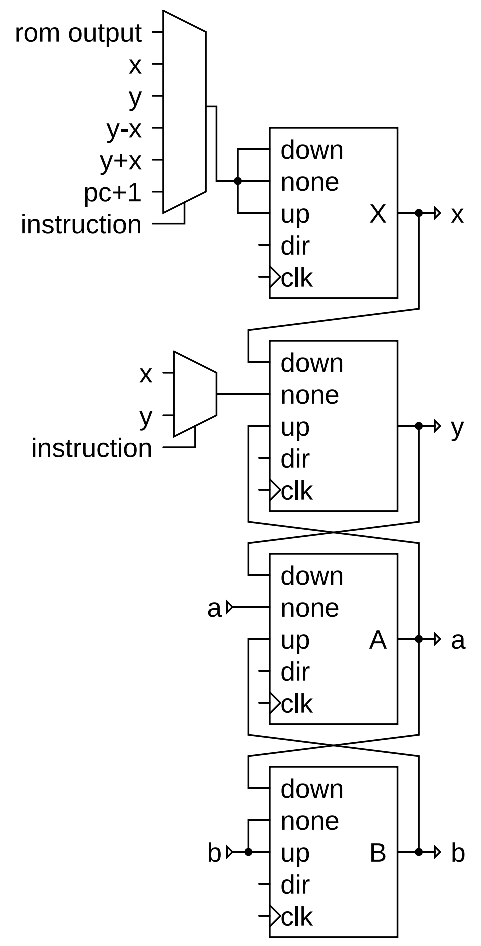
Opcode values from 0 to 127 (those with the most significant bit equal to 0) are all `PUSH` instructions and result in the instruction being pushed on the stack. Other opcodes should be assigned values corresponding to the number in the list above plus the last digit of your student number. For example, if your student number was 43211234, then the opcode for the `DUP` instruction should be $132 + 4 = 136$. Undefined opcodes should have no effect.

Each instruction causes the PC to be incremented unless the instruction explicitly loads it (JNZ, CALL, RET).

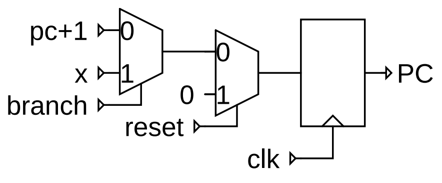
## Datapath Diagrams

The following (incomplete) diagrams show possible implementations for:

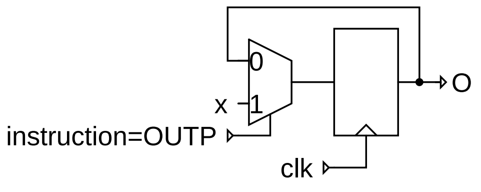
- the stack datapath:



- the PC datapath:



- the O datapath:



# Assignment - Part 1

## Data Types

Create a subtype of the `unsigned` data type, an 8-bit `byte` type. Use this data type throughout your design.

Define a constant for each mnemonic using the `byte` values appropriate for your student number.

Define a 2-bit subtype of `std_logic_vector`, `direction`, and define `direction` constants for the three possible stack movement directions: `down`, `none`, `up`[1].

Place all type and constant declarations in a package file, `cpu_package.vhd`.

## Components

Design and test two components: a stack register and a program memory.

The stack register should be declared as:

```
component stack_register is
        port ( reg_out : out byte ;
        if_down, if_hold, if_up : in byte ;
        dir : in direction ;
        clk : in std_logic ) ;
end component ;
```

which loads the register with the input selected by the `dir` input on the rising edge of `clk`.

The ROM component should be declared as:

```
component rom is
   port ( address : in byte ; data : out byte ) ;
end component ;
```

Write architectures for each of these components. These components will be used in Part 2.

Test each part of your design *separately*. As a minimum, demonstrate that:

- the memory gives the correct output for each program address input
- stack register loads the correct value for each of the possible stack movement directions.

[1]Normally this would be done with an enumerated type, but MaxPlus+II seems to have some bugs related to enumerated types.

The initial bytes of memory should contain the following program. This program generates on/off pulses whose duty cycle is controlled by the value on the input port.

## Program

```
;
; asg3.asm: assignment 3 assembly code for
; microcontroller
;
; EECE 379 1999/2000 Winter Session Term 2,
; Assignment 3
;
; Ed Casas, 2000/2/18
;
;
; Converts the value on the input port into a
; pulse-width modulated output on the LS bit
; of the output port.  The input value should
; be between 1 and 4.  If the input value is
; N, the output is on for approximately
; 12+4*(N-1) clock cycles and off for
; 15+4*(4-N) clock cycles.  The duty cycle of
; the output thus varies from 12/39 to 24/39
; and the period is always 39 clock cycles.

START   PUSH    1       ; turn the output on
        OUTP
        INP             ; read PWM value: 'ON' duration
        PUSH    DELAY   ; delay for 'ON' duration
        CALL
        PUSH    0       ; turn the output off
        OUTP
        PUSH    5       ; compute 'OFF' duration
        INP
        SUBT
        PUSH    DELAY   ; delay for 'OFF' duration
        CALL
        PUSH    START   ; repeat forever
        RET

; Time delay routine.  On entry expects N and
; the return address on the stack.  Both are
; popped.  Returns after 7+4*(N-1) cycles
; where N is the value on top of the stack.
; N must be >=1.

DELAY   SWAP            ; swap return address and count
DELAY1  PUSH    1       ; subtract 1 from count
        SUB
        PUSH    DELAY1  ; loop back if not done
        JNZ
        POP             ; pop (zero) count
        RET
```

Assemble the above program into your CPU's machine language and use it to design the instruction memory. Unused memory locations should be set to zero. You may use the opcode symbolic constants in your code.

For Part 1 of this assignment submit the VHDL code (3 files: `cpu_package.vhd`, `rom.vhd`, and `stack_register.vhd`) and the simulation results for the ROM memory and stack register (compile and test each one independently).

## Assignment - Part 2

Write a top-level entity that instantiates the components you designed and tested in Part 1 and completes the design by adding stack, PC and O datapaths and associated logic (e.g. to compute the values loaded into the X and Y registers, decode `PUSH` instructions, branch-decision logic and stack direction movement logic).

Your top-level entity should have three inputs: I, reset and clock, and three outputs displaying the current values of the O, PC and X registers. The PC and X outputs will allow you to monitor the execution of your program.

For Part 2 submit the VHDL code for *all* parts of your design and four pages of simulation results with a clock frequency of 1 MHz. Submit four pages of simulation results: two pages for the `I` input set to 1 and two for `I`= 4. The first page should show the output over the first 8 clock cycles and the second over all 50 clock cycles.

## Hints

The MaxPlus+II synthesizer occasionally issues "Unknown Error" messages due to compiler bugs. In this case you will have to remove (comment out) sections of the code to identify the error location and then try different alternatives until the error message disappears.

The compiler has other bugs. If a particular language feature does not seem to work as you expect, you may have to resort to a simpler alternative.