Z80 Microprocessor.

Inspect the subdirectory `tv80`. This design is an 8-bit Z80 microprocessor. The model includes a testbench that runs a small Z80 program on the microprocessor. You have to use Modelsim to answer questions about the behavior of the microprocessor as it executes this program.

Start by examining the testbench in the file `tb_tv80.v`. This model instantiates the Z80 processor, and connects it to a RAM. The RAM contains the opcodes of the program.

The following information will be useful to answer this question.

- The port definition of the Z80 is given in the following table.

Port	Width	Direction	Function
reset_n	1	input	Reset Input
clk	1	input	System Clock
wait_n	1	input	Wait State Request
int_n	1	input	Interrupt Request
nmi_n	1	input	Non-maskable Interrupt Request
busrq_n	1	input	Bus Request
di	8	input	Data Input
m1_n	1	output	M1 cycle (Start of Instruction)
mreq_n	1	output	Memory Request
iorq_n	1	output	IO Request
rd_n	1	output	Read Operation
wr_n	1	output	Write Operation
rfsh_n	1	output	Refresh Cycle
halt_n	1	output	CPU Halt State
busak_n	1	output	Bus Acknowledge
address	16	output	Bus Address
dout	8	output	Data Output

- The Z80 is a simple microprocessor. Every instruction takes a multiple of four clock cycles, so-called T-states. During the first cycle of every instruction, `m1_n` is asserted.

- After reset, execution starts at address `0x0`.

- The test program that executes on the Z80 is shown in the following listing. You can verify that the opcodes, shown on the left of the listing, correspond to the contents of the RAM memory used by the testbench.

```
0000                    ;; execution starts here
0000                    org     0x0
0000 26 00              ld      h, 0x0
0002 2e 3f              ld      l, 0x3f
0004 f9                 ld      sp, hl
0005            loop:
0005 2e 40              ld      l,0x40
0007 34                 inc     (hl)
0008 cb c6              set     0,(hl)
000a 18 f9              jr      loop
000c
000c                    ;; non-maskable interrupt
000c                    org     0x66
0066 e5                 push    hl
0067 2e 42              ld      l, 0x42
0069 34                 inc     (hl)
006a e1                 pop     hl
006b ed 4d              reti
006d
006d                    end
```

- The stack on the Z80 grows downward. The very first instructions initialize the stack pointer at `0x3F`.

- The procedure at address `0x66` is a non-maskable interrupt routine. This interrupt routine is triggered when the `nmi_n` input is asserted. As its name suggests, it is not possible to mask out the NMI request.

Question 1. Simulate the model, and find the contents of memory location `0x40` after `30,000ns`. This requires you to look carefully at the read/write operations in the memory during simulation. You may also inspect the Z80 program in order to infer how memory address `0x40` is used.

Answer: .....

Question 2. The testbench `tb_tv80.v` generates a non-maskable interrupt at `50,000ns`. This will cause the Z80 to execute the NMI interrupt service routine described in the listing. Find the number of clock cycles required to service the NMI. The service time is defined as follows.

- The *start of service* is the first upgoing clock edge after a low-to-high transition on `nmi_n`.
- The *end of service* is the first upgoing clock edge of the first machine cycle after completion of the NMI routine.

Answer: ..... `cycles`

Montgomery multiplication.

This design is an 8-bit Montgomery Multiplier, a component that is used in cryptographic hardware implementation. Inspect the design and its test-bench, and answers the following questions.

Question 1. Simulate the model, and find the number of clock cycles needed to complete one multiplication, as measures as a cycle distance from `start` to `done`.

Answer: . . . . . .

Question 2. Draw the architecture of the monmul module, as the result of RTL synthesis.

Answer: The answer comes in the form of a figure.

Question 3. Transform the design `monmul` into a faster `monmul2`. The second design must be twice as fast as the original one.

Answer: The answer is in the form of a new Verilog file.