YASAC Stages 1 and 2

Exercise 1. (Stage 1) Write a program that displays the number in *din* multiplied by 5. Test it by simulation. Modify the sample test bench as appropriate.

Exercise 2. (Stage 1) A nice way to test programs is to add a “$monitor” function call that prints interesting signals at every change. Add a “$monitor” to sample test bench that prints signals ‘start’, ‘reset’, ‘ready’, ‘din’ and ‘dout’. Use the test bench with the program in the previous exercise.

Exercise 3. (Stage 1 or 2) Assuming that the last two values in the Fibonacci sequence are stored in registers R0 and R1, write a program that display the next value of the sequence and updates R0 and R1. Modify the sample test bench as needed and simulate the program.

Exercise 4. (Stage 1 or 2) Implement the NOP instructions using the operation code 0. The NOP instruction does nothing. Write a simple program to test it by simulation. How can this instruction be useful?

Exercise 5. (Stage 2) Write a program that outputs to port01 the number in port08 multiplied by 5. Test it by simulation. Modify the sample test bench as appropriate.

Exercise 6. (Stage 1 or 2) Implement the ADDI and SUBI instructions. Write a simple program to test them and simulate it.

```
ADDI Ra, k  ; Ra ← Ra + k
SUBI Ra, k  ; Ra ← Ra - k
```

YASAC Stage 3

Exercise 7. Try to implement the stage 3 functionality in Verilog starting with the stage 2 code and using the specifications in the slides. Working in groups is encouraged.

Exercise 8. Write a test program that, starting at $n_0 = 0$, generates the sequence $n_i = i*k$ at port01, where k is the value in port08 at the time the program starts executing.

E.g., if port08=7, the sequence generated at port01 should be:

0, 7, 14, 21, 28, ...

Exercise 9. Write a program that shows a countdown from 10 to 0 in port01. At the end of the countdown it should output the number 0x8888 in port02:port01.

Exercise 10. Instructions CP (ComPare) and CPI (ComPare with Immediate) are identical to SUB and SUBI, except the former do not update the destination register, only set flags in the status register. These are very useful to compare numbers and take decisions using branching instructions.

a) Implement the CP and CPI instruction.

b) Write a test program and simulate it.

YASAC Stage 4

Exercise 11. Try to implement the stage 4 functionality in Verilog starting with the stage 3 code and using the specifications in the slides. Working in groups is encouraged.

Exercise 12. A number in port08 represents the ASCII code of a character. Write a program that determines if the code corresponds to a letter. In that case, write the code of the upper case letter to port02. In other case, write 0 to port02. In either case, write the input code to port01.

Exercise 13. Write a program to count the number of bits set to 1 in port08. Save the number to port01.

Exercise 14. A number in port08 is encoded in packed natural BCD format (it holds two BCD digits, each digit represented in base 2). Write a program that multiplies the number by 2 and saves the result (in packed natural BCD format) in port02:port01 (the combination of the two ports).

Exercise 15. Implement the NEG instruction. Write a test program and simulate it.

```
NEG Ra  ; Ra ← -Ra. Flags activated like in 0 – Ra.
```

Exercise 16. Many processors have an Arithmetic Shift Right (ASR) instruction that shift bits to the right and fills in using the sign bit. E.g. ASR(10010101) = 11001010, ASR(01101010) = 00110101.

a) Explain why this instruction is useful and the reason for its name.

b) Write a sequence of YASAC instructions equivalent to the ASR.

If you think you may need the ASR instruction in the future, you can implement it in the YASAC, just for fun.

YASAC Stage 5

Exercise 17. Write a subroutine that multiplies unsigned numbers in registers R1 and R2 and returns the result in register R1. Do not take into account possible overflow. Use the subroutine in a program that reads a number in port08 and puts the result from multiplying this number by 7 in port01.

Exercise 18. Write a blinking LED test program. The program should make an LED connected to bit 0 of port03 to blink once per second (changes its value once every ½ second). Assume the computer's clock signal frequency is 1000Hz. Use subroutines to optimize the code when possible (e.g. a subroutine that waits for half a second and a subroutine that complements bit 0 of port03 may be useful here).

Exercise 19. A very useful addressing mode for compilers is the register indirect addressing mode with displacement. It is specially useful to handle the **stack frame** used to implement local variables in higher-level programming languages[1]. This addressing mode may be implemented in the YASAC with the following instructions:

```
LDD Ra, Rb+q      ; Ra ← datamem(Rb+q)
STD Rb+q, Ra      ; datamem(Rb+q) ← Ra
```

These instructions would substitute the current LD and ST since these become a particular case of LDD and STD with no displacement (q=0).

a) Propose an instruction format for the new instructions. Note that the displacement may take less than 8 bits.

b) Propose the necessary changes in the YASAC 5 data unit in order to implement the LDD and STD instructions.

c) Calculate the sequence of micro-operations that are necessary to execute these new instructions.

1 https://en.wikipedia.org/wiki/Call_stack#STACK-FRAME

Extra exercises ideas and small projects

Project: simple alarm program.

Project: gate control program.

Project: program an equivalent to the simple or the RPN calculator of unit 3 in YASAC. Test it.

Rationale

-