

**ECE 366—Fall 2001, Instructor: Prof. Shantanu Dutt**

**Processor Design Project I**

Due date : Fri Oct. 12

**Note:** You need to do this project in teams of two. Please submit by Wed. Oct. 3 the name and ID of your team to the TA (crangasw@ece.uic.edu).

1. Design a C.U. for the myth8 using a control program that implements the following instructions:

1. [nop]

Semantics: waste a cc and go back to the fetch state

nop opcode = 000000 (all other instructions fields are also 0).

2. [add  $r_i$   $r_j$   $r_k$ ] – this will be type 1

Semantics:  $r_i \leftarrow r_j + r_k$

add opcode = 000001

3. [add\_imm  $r_i$   $r_j$  blank 4-bit constant] – type 1 ( $r_k$  field is blank)

Semantics:  $r_i \leftarrow r_j + 4\text{-bit constant}$

add\_imm opcode = 000011

4. [sub  $r_i$   $r_j$   $r_k$ ] – type 1

Semantics:  $r_i \leftarrow r_j - r_k$

sub opcode = 000100

5. [rolm  $r_i$   $r_i$   $r_i$  4-bit constant] – type 1

Semantics:  $r_i$  is rotated left (note that MSB goes into LSB) as many times as specified in the 4-bit constant field. Assume that this 4-bit constant is  $\geq 0$ .

rolm opcode = 000101

6. [bge blank  $r_j$   $r_k$   $rel\_addr$ ] – type 1 ( $r_i$  field is blank)

Semantics: if  $r_j \geq r_k$ ,  $r_7 \leftarrow r_7 + rel\_addr$

blt opcode = 000110

7. [rorm  $r_i$   $r_i$   $r_i$  4-bit constant] – type 1

Semantics:  $r_i$  is rotated right (note that LSB goes into MSB) as many times as specified in the 4-bit constant field. Assume that this 4-bit constant is  $\geq 0$ .

rolm opcode = 000111

**Hint:** For n-bit registers, a k-bit rotate right is the same as an  $(n - k)$ -bit rotate left,  $k \leq n$ .

8. [jmp 8-bit address] – type 2

Semantics:  $r_7 \leftarrow 8\text{-bit address}$

jmp opcode = 001000

9. [load\_imm  $r_i$  8-bit constant] – this will be a type 2 instruction.

Semantics:  $r_i \leftarrow 8\text{-bit constant}$

load\_imm opcode = 001001

10. [load  $r_i$  8-bit address] – type 2

Semantics:  $r_i \leftarrow \text{MEM}[8\text{-bit address}]$

load opcode = 001010

11. [load\_ind  $r_i$  ( $r_j$ )] – type 1. This is a **load word** instruction using the *register-indirect* addressing mode.

Semantics:  $r_i \leftarrow \text{Mem}[r_j]$

load\_ind opcode = 001011

12. *store* is a 24-bit instruction of the form

1st 16 bits (least significant 2 bytes): [store blank  $r_j$   $r_j$ ]

3rd (most significant) byte: [8-bit address]

Semantics:  $\text{MEM}[8\text{-bit address}] \leftarrow r_j$ . Note that this instruction does not fall into either type 1 or type 2 as described in the myth 8 manual. However, that does not mean that we cannot design the CU for other instruction formats. We can do it as long as the hardware allows us to process such instructions. There are no hard and fast rules in architecture design and in digital design in general; there are lots of design options and flavors!

store opcode = 001100

13. [sw+  $r_j$   $r_j$   $r_k$  4-bit-offset  $X$ ] –type1. This is a **store word** instruction using the *auto-increment register-indirect* addressing mode. Semantics:  $\text{Mem}[r_j] \leftarrow r_k$ ;  $r_j \leftarrow r_j + X$ . Note that  $r_j$  appears in both the  $r_i$  and  $r_j$  fields of the instruction.

sw+ opcode = 001101

14. [halt]

Semantics: Stop the program.

halt opcode = 001110

**Important Note:** The C.U. should not use the the user registers (those specified by the  $r_j$  and  $r_k$  fields) specified by the instruction and the PC ( $r_7$ ) as scratch-pad registers. Only  $r_4$  to  $r_6$  can be used as scratch-pad registers.

Also, debug your design using sample instructions that you put in a .mem file.

2. On the date submit your debugged and tested control program. Theoretically analyze the # of cc's for executing each instruction (assume memory read/write takes 3 cc's). (show your work clearly) and in a tabular form.

You will be given one or two machine language programs (.mem files) to test your C.U. design. Report the initial and final states of the mythesim graphical interface (with all register values etc.) and the number of clock cycles required to execute each program.

3. **Grading criteria:**

- (a) **Correctness: 60%** of the project points. Your CU design should be able to execute the given programs as well as a “blind” program (that you will not have access to) **correctly** to get anywhere near full points. So make sure that you have debugged the execution of each instruction carefully with relevant different values in the “variable” fields ( $r_i, r_j, r_k$ , 4-bit constant, 8-bit constant, etc.). You cannot of course exhaustively debug for all possible values in the variable fields, but use different classes of values (e.g., in the rolm instruction, test it for two different value of  $r_i$ , for a few different values stored in the registers (e.g, values that will not change: 0, -128, and values that will change 127, 64, -40, etc.), and for the following values of the *4-bit const.*: positive boundary values 0000, 0111, one or two interior values like 0100). Relevant values of the variable fields for debugging instructions generally depend on the instruction.

The bulk of the points (**60%**) will be for correctness, i.e., if your design is correct (determined by running programs correctly [**40%**] **and** by manual checking of your .uicode file [**20%**] you get **60%** of the points. **However, if your design is incorrect (i.e., the .mem programs do not run correctly) you will get no more that 10% of the points for this project.**

- (b) **Speed:** A smaller portion (**20%**) of the points will be reserved for how fast the programs run for your CU design (matters only if they run corectly!).
- (c) **Number of states:** An even smaller portion (**10%**) will be reserved for total number of states—this represents hardware cost (matters only if design is correct!).
- (d) **Clock cycle analysis of instructions: 10%** of project points.

4. What to submit (only one set per team):

- (a) Electronic submission of your .uicode file using the *turnin* command (will give details on web page later).
- (b) Hardcopy of: (i) clock cycle analysis; (ii) Initial and final states of the mythesim graphical interface (with all register values etc.) and the number of clock cycles required to execute each program—use screen capture for this; (iii) .uicode file;

5. Total points for this project:

**40% of total project weight**