SAN JOSÉ STATE UNIVERSITY
Charles W. Davidson College of Engineering
DEPARTMENT OF ELECTRICAL ENGINEERING

EE271 Final Design Project Assignment
A Simple 8-bit Scalar Processor

The purpose of this project is to design, optimize and analyze a simple 8-bit scalar processor. All your work including the details of your design, implementation, tests, measurements, analysis, design options, EDA tool setting and configurations must be clearly described as stated in the project report template.

I. Design Specifications

I.1 User Visible Registers:

- 1 8-bit data register DR
- 1 8-bit general register GR
- 1 8-bit address register AR
- 1 8-bit program counter register PR
- 1 4-bit flag register FR

Except flag register, all other user visible registers can be used as a source register (S) or a destination register (D)

I.2 Design Limitations

- All instruction machine codes are 8-bit codes
- Memory address is always stored in address register (AR)
- 4-bit flag register (FR) stores 4 1-bit status flags, in order from MSb to LSb are the zero flag (ZF), signed flag (SF), carry flag (CF), and overflow flag (OF)
- Processor can access 256 bytes of external memory, which include
  - 128 bytes code memory with addresses from 0 to 127
  - 128 bytes data memory with addresses from 128 to 255

I.3 Processor Interface: The processor interface ports must be exactly as below.

- `rst` Active-high reset input signal (reset all registers to 0)
- `clk` Clock input signal
- `add[7:0]` 8-bit output address bus
- `dat[7:0]` 8-bit bidirectional data bus
- `wrt` Active-high memory write control signal
- `rd` Active-high memory read control signal

I.4 Operation Order:

1. Fetching an instruction from memory
2. Updating the program counter register PR
3. Decoding the instruction
4. Executing the instruction

5. Back to step 1

I.5 Available Instructions and Operands

- NOP: No operation
- JMP: Branch the activity (jump to other instruction in memory) by loading the program counter register PR with value stored in address register AR
- CPR D, S: Copy the content of source register specified in the instruction (S) to destination register specified in the instruction (D)
- LLS Load the least-significant 4 bits of this machine code to LEAST-significant half of general register GR
- LMS Load the least-significant 4 bits of this machine code to MOST-significant half of general register GR
- CFR Copy the contents of 4-bit flag register FR to the LEAST-significant half of the general register GR
- RDM D: Read a byte from memory location specified by the address register AR into destination register specified in the instruction (D)
- WRM S: Write the contents of the source register specified in the instruction (S) into memory location specified by the address register AR
- ADD D, S: Add the contents of the source register specified in the instruction (S) to destination register specified in the instruction (D). The processor uses 2's complementary number system for arithmetic operations
- SUB D, S: Subtract the content of the source register specified in the instruction (S) from destination register specified in the instruction (D). The processor uses 2's complementary number system for arithmetic operations

When power-on, all registers are reset to zero, including the program counter register PR. This means that the first instruction should be at the memory address 00H.

The operational codes of the instructions and binary codes of the register are as below. Unused bits are treated as don't care.

Operational Codes	
NOP	0000
JMP	0001
RDM	0010
WRM	0011
CPR	0100

Operational Codes	
ADD	0101
SUB	0110
LLS	0111
LMS	1000
CFR	1001

Registers	
AR	00
DR	01
GR	10
PR	11

Below are examples of some instructions and machines codes

Instruction	Machine code	Explanations
JMP	0001xxxx	Load the content of AR to PR
WRM DR	001101xx	Write data in DR to a memory location specified by AR
ADD GR,DR	01011001	Add the data in DR and GR and store the result in GR
LLS	01111100	Load value 12 (1100) to the least-significant half of GR

II. Library and EDA Tools

- Use Toshiba library at **`/export/apps/toshiba/sjsu/synopsys/tc240c/`** for your synthesis and analysis. This is a technology library with 250 nm technology operating at 2.5V. For best and worst delays (hold and setup time violation checks), use **`tc240c.db_BCCOM25`** and **`tc240c.db_WCCOM25`**, respectively. Note that in using Toshiba library, the `tc240c.workview.sdb` is the symbol library that you should use.
- RTL-level simulations can be performed on any simulator but gate-level (netlist) simulations must be performed with either **Synopsys VCS** or **Cadence NCVERILOG**. Synthesis must be performed with **Synopsys Design Vision** (or **Design Compiler**)
- All simulations must be performed at both RTL and gate (netlist) levels

III. Design Implementation and Testing

III.1 Number of Clock Cycles per Operation

To simplify the design, use 1 clock cycle as the operation time for NOP instruction and 2 clock cycles for JMP, LLS, LMS, CFR, CPR instructions. The memory instructions (RDM and WRM) and machine code fetching are performed by two-step process: the address activation and the data access. Each step requires a number of clock cycles (one or more than one) and students are required to do some analysis/study/experiments to estimate the number of clock cycles for these operations. Similarly, the relative delays of the ADD and SUB instructions need to be measured and analyzed in order to define the optimum number of clock cycles for these instructions. Not optimizing the number of clock cycles for the memory and arithmetic instructions may result in bad processor performance.

III.2 Required Modular Test Cases and Final Test Case

Each module and each instruction need to be tested during the project development as specified in the report template. Students are also required to develop a final comprehensive test case to test the processor operation. This final test cased will be used to test the design at both RTL and gate (netlist) level. The outline for the final test case should be developed based on the test procedure as listed below. Note that you can keep the processor idling in a time period by keeping the reset signal active.

- Develop a completed sequence of machine codes for the test case
- Develop test case data for the above sequence of machine codes
- Load the machine codes into the memory, starting at memory address 00H
- Load the test data into memory, starting at memory address 80H
- The processor starts to execute the final comprehensive test case once the machine codes and test data are available in the memory.

The final comprehensive test case performs operations as outlined below. Students are required to use the `$display` system task in the testbench to display the data and results at the defined time steps in order to observe the correctness of the processor operations. Below is the program operation. Note that the flags must be 1 or 0 depend on the status of the arithmetic operations `ADD` and `SUB`.

1. Add an 8-bit data at memory location 80H to an immediate 8-bit data 45H and store the 8-bit result to memory location 80H
2. Add 2 8-bit data at memory locations 81H and 91H and store the 8-bit result to memory location 81H
3. Subtract an 8-bit data at memory location 82H from another 8-bit data at memory location 92H and store the 8-bit result to memory location 82H
4. Use `JMP` instruction to repeat steps #1 through #3 one more time
5. Store the 4-bit flag register into the least-significant 4 bits of memory location F0H

III.3 Gate-Level Testing

Students must run the final test case at both the RTL-level and gate-level (netlist). The final test case must be developed such that it can be used for both RTL simulations (pre-synthesis) and logic (netlist) simulations (post-synthesis simulation.) without modification.

Some notes:

In order to run VCS with `netlist` *file, you need to point to the Verilog source of the target library. As an example of synthesizing/mapping with Toshiba library, then the VCS command for simulating your 16-bit adder netlist named* `add_16_tc240c_netlist` *should be as below:*

```
vcs +v2k –y /export/apps/toshiba/sjsu/verilog/tc240c
+libext+.tsbvlibp add_16_TB.v add_16_tc240c_netlist.v
```

Moreover, you can use Cadence simulator (`ncverilog`) as below:

```
ncverilog -y /export/apps/toshiba/sjsu/verilog/tc240c
+libext+.tsbvlibp +access+r add_16_TB.v
add_16_tc240c_netlist.v
```

where `.tsbvlibp` is the suffix of Verilog source files of `tc240c` library and `/export/apps/toshiba/sjsu/verilog/tc240c` is the library directory

Since tc240c library has nanosecond time scale and 10 picosecond time-precision, the timescale directive below should be included in the Verilog testbenches

```
`timescale 1 ns / 10 ps
```

Note that some Verilog files in this technology library are encrypted (protected) by Cadence EDA tool and VCS may not be able to decrypt them. If that is the case, you will then get error messages. Using `ncverilog` for netlist simulation should be fine since the library was provided to us by Cadence. If anyway you get error message about file protection, try to find out in your netlist which encrypted Verilog files that are used and then try to replace them with non-encrypted files. Below is the list of encrypted files.

- `tsbLD2SFprim.tsbvlibp`
- `tsbSCK2prim.tsbvlibp`
- `tsbMUXXprim.tsbvlibp`
- `tsbSCK1prim.tsbvlibp`

- `tsbCTLprim.tsbvlibp`
- `tsbFD2BASICprim.tsbvlibp`
- `tsbFD4BASICprim.tsbvlibp`
- `tsbCHKprim.tsbvlibp`
- `tsbRCK1prim.tsbvlibp`
- `tsbCLD3SFprim.tsbvlibp`
- `tsbCLD4SFprim.tsbvlibp`
- `tsbMAJprim.tsbvlibp`
- `tsbLDP1prim.tsbvlibp`
- `tsbFD1BASICprim.tsbvlibp`
- `tsbCFD3BASICprim.tsbvlibp`

IV. Design Project Report and Report Submission

EE271 final project is an individual project, each student will work on the project separately and no information should be shared among students. Each student is required to turn-in a CDROM or memory stick that includes a formal "Final Project Report" (SOFT-COPY) with a required format (a template report is provided) and the whole project materials such as Verilog codes, netlist, simulation waveforms, circuits and reports from the simulation and synthesis tools. Students are responsible for providing completed information such that grader can re-simulate and re-synthesize the design for grading purposes.

The CDROM or memory stick must have one file named "`Report.xxx`", one file named "`README.xxx`", and one directory named "`Project`". The "`Report.xxx`" file is the final project report in MSWORD or PDF, the "`README.xxx`" file explains steps to simulate and synthesize your project, and the "`Project`" directory is the parent directory that contains files, scripts, outputs, etc. of the project. Files and sub-directories under the parent directory "`Project`" must be organized as they are on your computer account such that grader can just copy the whole `Project` structure to his/her computer account and will be able to simulate and synthesize your project correctly. Please label directly on the CDROM or memory stick your exam seat #, your full name and your phone number.

The project report must be in the right format and must include completed information about the project as shown in the report template. Please download the report template on the class canvas and edit your report directly from the sample. Do not change the format, fonts, and page setup of the sample report but just fill-in and writing the requested information. Do NOT skip the information and do NOT report extra information. Grading will be based on the quality and completeness of your project and report.

Write the information below directly on your CDROM or memory stick, NOT on the cover or the envelope, so that grader can contact you if there is any problem with your turn-in.

- **EE271 Fall 2014**
- **Your exam seat number, full name, and phone number**