VLSI System Design (Graduate Level)
Fall 2010

HOMEWORK III
Due day: midnight December. 13, 2010

In this homework, you are to complete the design of a simplified pipeline CPU (at least 3 stages), referred to as mini-pipeline CPU (mPC). It could be a CPU derived from SISC taught in class or developed by you own. The most distinct feature of mPC is its basic hazard resolution capability. Your mPC must meet the specified requirements.

There will be bonus points for doing extra work for those who like challenges. Partial credits will be given based on level of completeness for bonus parts. Points for this homework are depicted as follows.

Baseline (300 points): Complete a working synthesized CPU
**Bonus-I** (+150 points): Complete a working synthesized CPU with a cache
**Bonus-II** (+250 points): Complete a working synthesized CPU with a cache and interrupt (or exception) mechanism

**NOTE:** Suppose you complete a working synthesized CPU with a cache, your grade will be the summation of baseline + Bonus-I, i.e., 450 points. You only need to deliver one set of code and verification. Therefore, you shall specify in your report what you have achieved.

## General rules for deliverables

- This homework needs to be completed by INDIVIDUAL student.
- Compress all files described in the problem statements into one zip or rar.
- Submit the compress file to the course website before the due day. **Warning!** AVOID submit in the last minute. Late submission is not accepted.

## Grading Notes

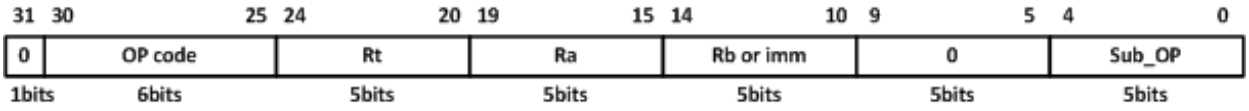
- **Important!** DO remember to include your Verilog code. NO code, NO grades. Also, if your code can not be recompiled by TA successfully using tools in SoC Lab, you receive NO credit.
- Write your report seriously and professionally. Incomplete description and information will reduce your chances to get more credits.
- If extra works (like synthesis, post-simulation or additional instructions) are done, please describe them in your final report clearly for bonus points.
- Please follow course policy.

HOMEWORK III

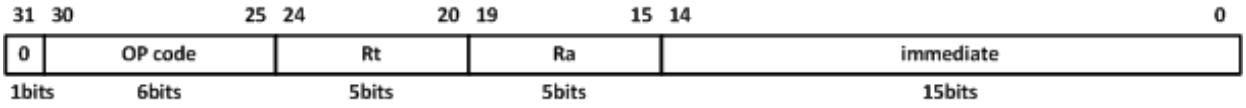
# Exercise

**mPC** has the following instruction format basically

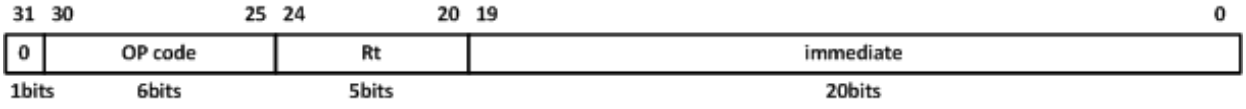
☞Data-processing (SE:sign-extened, ZE:zero-extened)




| OP code | Mnemonics | DST | SRC1 | SRC2 | Sub OP | Description |
|---|---|---|---|---|---|---|
| **100000** | NOP | 00000 | 00000 | 00000 | 01001 | No operation |
| **100000** | ADD | $Rt | $Ra | $Rb | 00000 | Rt = Ra + Rb |
| **100000** | SUB | $Rt | $Ra | $Rb | 00001 | Rt = Ra - Rb |
| **100000** | AND | $Rt | $Ra | $Rb | 00010 | Rt = Ra & Rb |
| **100000** | OR | $Rt | $Ra | $Rb | 00100 | Rt = Ra \| Rb |
| **100000** | XOR | $Rt | $Ra | $Rb | 00011 | Rt = Ra ^ Rb |
| **100000** | SRLI | $Rt | $Ra | imm | 01001 | Shift right : Rt = Ra >> imm |
| **100000** | SLLI | $Rt | $Ra | imm | 01000 | Shift left : Rt = Ra << imm |
| **100000** | ROTRI | $Rt | $Ra | imm | 01011 | Rotate right : Rt = Ra >> imm |




| OP code | Mnemonics | DST | SRC1 | SRC2 | Description |
|---|---|---|---|---|---|
| **101000** | ADDI | $Rt | $Ra | imm | Rt = Ra + SE(imm) |
| **101100** | ORI | $Rt | $Ra | imm | Rt = Ra \| ZE(imm) |
| **101011** | XORI | $Rt | $Ra | imm | Rt = Ra ^ ZE(imm) |
| **000010** | LWI | $Rt | $Ra | imm | Rt = Mem[Ra + (imm << 2)] |
| **001010** | SWI | $Rt | $Ra | imm | Mem[Ra + (imm << 2)] = Rt |

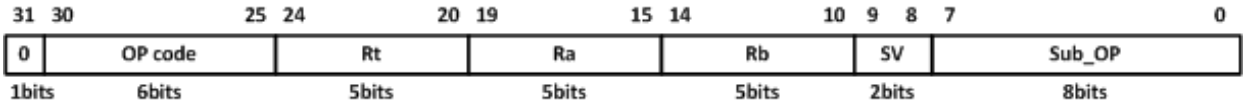



| OP code | Mnemonics | DST | SRC1 | Description |
|---|---|---|---|---|
| **100010** | MOVI | $Rt | imm | Rt = SE(immediate) |

VLSI System Design (Graduate Level)
Fall 2010

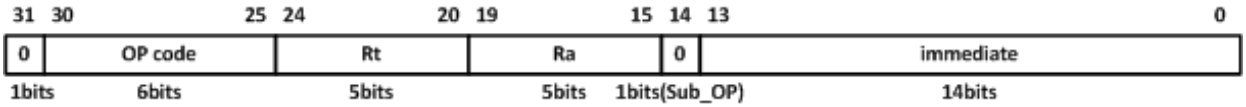
HOMEWORK III

☞Load and store



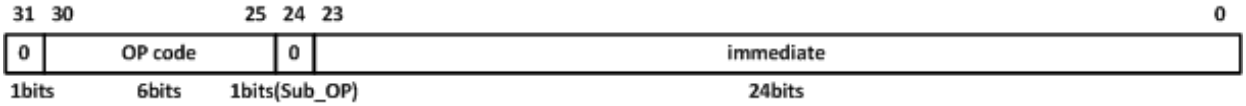

| OP code | Mnemonics | DST | SRC1 | SRC2 | Sub OP | Description |
|---|---|---|---|---|---|---|
| **011100** | LW | $Rt | $Ra | $Rb | 00000010 | Rt = Mem[Ra + (Rb << sv)] |
| **011100** | SW | $Rt | $Ra | $Rb | 00001010 | Mem[Ra + (Rb << sv)] = Rt |

☞Branch (SE:sign-extened)




| OP code | Mnemonics | DST | SRC1 | Sub OP | Description |
|---|---|---|---|---|---|
| **100110** | BEQ | $Rt | $Ra | 0 | if(Rt == Ra) PC = PC + SE(imm << 1)<br>else PC = PC + 4 |

☞Jump (SE:sign-extened)




| OP code | Mnemonics | Sub OP | Description |
|---|---|---|---|
| **100100** | J | 0 | PC = PC + SE(imm << 1) |

**Baseline** (300 points) Design a pipeline CPU (either extending the SISC as the baseline design or developing by your own) with the following features:

a. Nineteen instructions equivalent to above basically
b. Bypassing mechanism to resolve the interlock problem
c. Data and instruction memory has zero delay
d. No need to synthesize the data and instruction memory
e. Revise the example of the constraint file given in HWII(Part2) Problem1 to do synthesis (Note that the output load shall not be changed!!)
f. Verification
   i. Perform verification for individual instruction
   ii. Develop 5 distinct testbenches for hazard problems
   iii. Read a data array, whose size is 10, from data memory and restore them back with the sorted order in the same data array. Please give 3 examples to show that the machine code is working fine.

iv. Design a Fibonacci sequence generator that can give at least first 10, i.e., $F_{0...}F_{10}$. In math term, $F_n = F_{n-1} + F_{n-2}$, where $F_0$=0 and $F_1$=1.

**Bonus I** (+150 points) Extend the synthesizable pipeline CPU of HW3 Problem1 to equip with a cache. Note that the same verification in HW3 Problem 1 shall be applied and make them work.

**Bonus II** (+250 points) Extend the pipeline CPU to equip with a cache and interrupt. Note that the same verification in HW3 Problem 1 shall be applied and make them work.

# Deliverables

1. All CPU Verilog codes including components, testbenches and machine code for each lab exercise.
2. A homework report that includes
    a. A summary in the beginning to state what has been done (such as baseline mPC, synthesis, post-synthesis simulation)
    b. A block diagram for your completed mPC indicating all necessary components and I/O pins
    c. Instruction set, instruction format, and addressing modes
    d. Simulated waveforms for at least required test benches with proper explanation
    e. Report your maximum simulation speed
    f. Synthesis reports and log file
    g. Verification of post-synthesis simulation waveforms with proper explanation
    h. Conclusion

HOMEWORK III

////////////////////////////////////////////////////// *Reference* //////////////////////////////////////////////////////

☞SMILE CPU has the following instruction format (Andes ISA)

The Andes 32bit instruction formats and the meaning of each filed are described below:

- Type-0 Instruction Format

| | | |
|---|---|---|
| 0 | Opc_6 | {sub_1, imm_24} |

- Type-1 Instruction Format

| | | | | |
|---|---|---|---|---|
| 0 | Opc_6 | rt_5 | imm_20 | |
| 0 | Opc_6 | rt_5 | sub_4 | imm_16 |

- Type-2 Instruction Format

| | | | | |
|---|---|---|---|---|
| 0 | Opc_6 | rt_5 | ra_5 | imm_15 |
| 0 | Opc_6 | rt_5 | ra_5 | {sub_1, imm_14} |

- Type-3 Instruction Format

| | | | | | |
|---|---|---|---|---|---|
| 0 | Opc_6 | rt_5 | ra_5 | rb_5 | sub_10 |
| 0 | Opc_6 | rt_5 | ra_5 | imm_5 | sub_10 |

- Type-4 Instruction Format

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | Opc_6 | rt_5 | ra_5 | rb_5 | rd_5 | sub_5 |
| 0 | Opc_6 | rt_5 | ra_5 | imm1_5 | imm2_5 | sub_5 |

- opc_6: 6-bit opcode
- rt_5: target register in 5-bit index register set
- ra_5: source register in 5-bit index register set
- rb_5: source register in 5-bit index register set
- rd_5: destination register in 5-bit index register set
- sub_10: 10-bit sub-opcode
- sub_5: 5-bit sub-opcode
- sub_4: 4-bit sub-opcode
- sub_1: 1-bit sub-opcode
- imm_24: 24-bit immediate value, for unconditional jump instructions (J, JAL). The
- immediate value is used as the lower 24-bit offset of same 32MB memory block (new
- PC[31:0] = {current PC[31:25], imm_24, 1’b0}
- imm_20: 20-bit immediate value. Sign-extended to 32-bit for MOVI operations.
- imm_16: signed PC relative address displacement for branch instructions.
- imm_15: 15-bit immediate value. Zero extended to 32-bit for unsigned operations,
- while sign extended to 32-bit for signed operations.
- imm_14: signed PC relative address displacement for branch instructions.
- imm_5, imm1_5, imm2_5: 5-bit unsigned count value or index value

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////