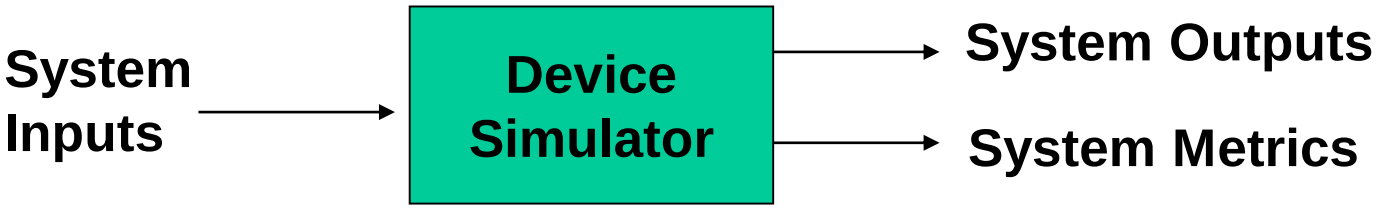
CPU function and performance simulator

A Computer Architecture Simulator

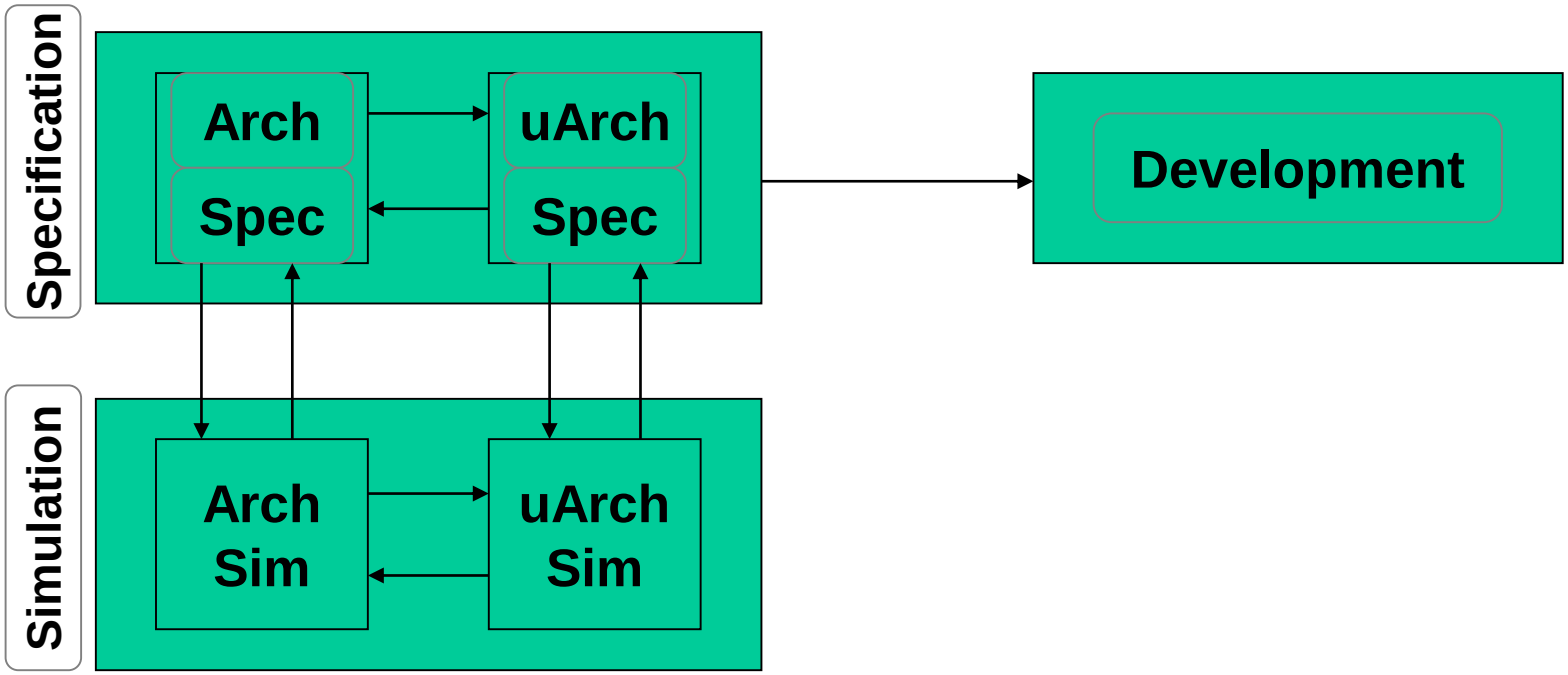
- **What is an architectural simulator?**
  - **A tool that reproduces the behavior of a computing device**



- **Why use a simulator?**
  - **Permits more design space exploration**
  - **Facilitates validation before H/W becomes available**
  - **Level of abstraction can be throttled to design task**
  - **Possible to increase/improve system instrumentation**

(originated from simplescalar hacker guide)

Functional vs. Performance Simulators

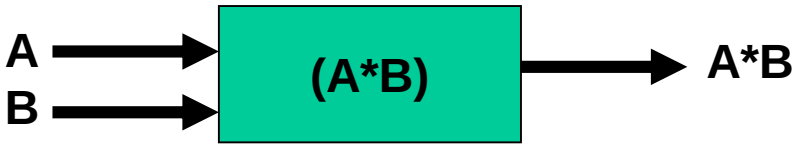


- **Functional simulators implement the architecture**
  - The architecture is what programmer's see
- **Performance simulators implement the detailed behavior of the hardware**
  - Model system internals (microarchitecture)
  - Often concerned with time

(originated from simplescalar hacker guide)

Functional vs. Performance Simulators (Cont'd.)

- **For example:**
  - Functional simulators
    - Simulate the functionality of the processor only



  - Performance simulators
    - Simulate the functionality and the detailed hardware behavior

Goal of this Project

- **Utilizing high-level programming language, such as C, to perform detailed CPU simulation**
- **You will build the whole CPU simulator step by step**

Steps of Building the Simulator

- **Functional simulation**
- **Performance simulation**
  - single-cycle datapath
  - Multi-cycle datapath
  - Pipeline

Functional Simulation

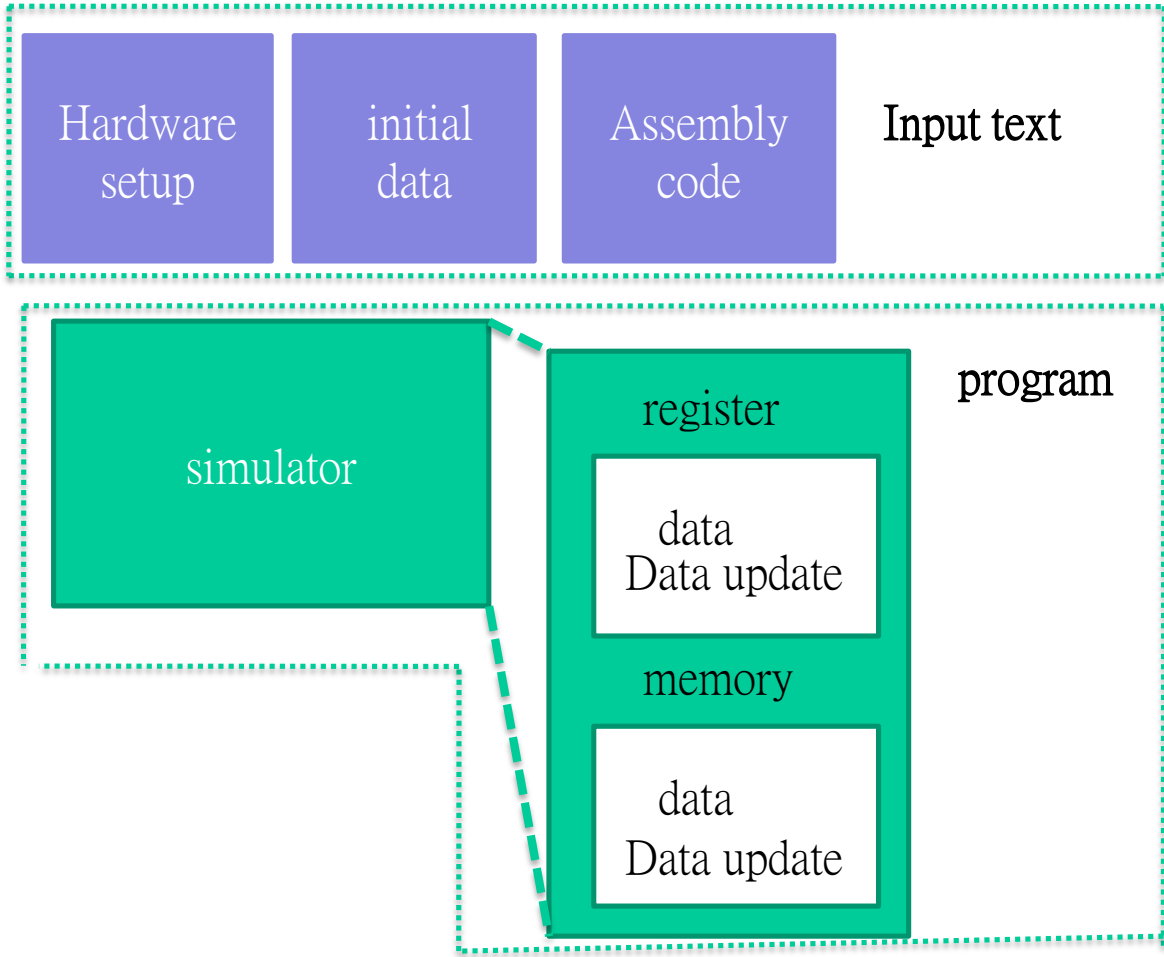
- **All functions of a CPU are defined by its Instruction Set**
- **Functional Simulation = Perform functional simulation of each instruction**
  - E.g. instruction “add $1, $2, $3”
    its C language simulation $1 = $2+ $3;

Today's Goal

- **Go through the code structure**
- **Implement the functional simulation of 7 instructions**

Overview of simulator

- **模擬流程**
  - 設定register大小、memory大小、datapath type
  - 設定系統初始值
  - 輸入要執行的instruction
  - 執行instruction
  - 輸出simulation結果



Input/Output of the Simulator

- **Input**
  - Hardware configuration
    - Specified in the HW_Config.cfg file
  - Initial data placement
    - Data memory & register
    - Specified in the init_data_placement.txt
  - Assembly code
    - Specified in the assembly_code.txt
- **Output**
  - Includes two types of results
    - Performance results
      - # of CPU cycles, cycle time, etc.
    - The contents of the data memory and register file
  - is specified in “perf_results.txt ”

Structure of the Code

- **Starting from main.c**
- **The simulator has 3 major components**
  - Setting up the hardware configuration of the target CPU architecture
    - Defined in machine.h
  - Simulation of instruction behavior
    - Defined in inst_process.h
  - Simulation of datapath
    - Defined in sim.h
    - Simulate the behavior of each datapath

Usage of the Code-Linux environment

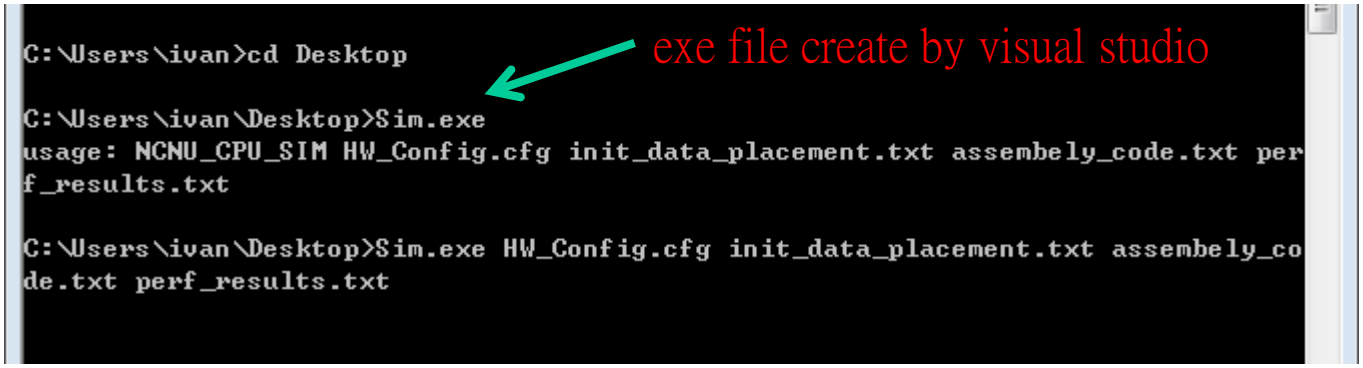
- **To compile the code**
  - make
- **To remove all the compiled objects**
  - make clean
- **executable after the compilation**
  - NCNU_CPU_SIM
- **To run the simulator**
  - ./NCNU_CPU_SIM HW_Config.cfg init_data_placement.txt assembely_code.txt perf_results.txt

作業說明

- **步驟**
  - 將範例程式缺少的部分填寫完成
  - 執行程式並用simulator run範例assembly code
  - 觀看dump 出來的 register 與 memory 內容是否正確
- **須完成以下指令的功能模擬**
  - add, sub, addi, subi, or, and, sll, srl, lw, sw, slt, beq
  - 會公布測試assembly code 給同學測試, demo時除了測試code以外, 還會隨機測試另外一組code

Usage of the Code-Windows environment

- **Open a project and include code into project**
- **Compile the code**
- **To run the simulator**
  - Open cmd.exe and execute the program



Machine.h

- **Defines the**
  - Parameters & data structure utilized in the hardware configuration
  - Functions for initializing the hardware configuration

HW_Config.cfg

- **Specifying the hardware configuration**
- **This file is processed by void setup_hardware_cfg() in machine.c**

ecns1.csie.ntu.edu.tw - PuTTY

```
    I    hw.cfg
# defines the hardware configuration of the target system
# words after the # sign is for comments
# - indicates hardware parameters

-datapath_type  SINGLE  # indicates the type of datapath, can be SINGLE, MULTI, and PIPELINE

-data_mem_size  2048    # in bytes

-inst_mem_size  2048    # in bytes

-reg_size       16      # in num. of words

-op_mem_lat     100     # in pico seconds

-op_ALU_lat     30      # in pico seconds

-op_reg_lat     30      # in pico seconds
```

Data Structure of Memory

- **Both instruction & data memories utilize this data structure**
- **With 3 fields**
  - Address: must be a multiplication of 4
  - Data: an array of char type
  - Label: an array of char type
    - Only be utilized in the instruction memory
- **NOTE**
  - Instruction and data memories share the same address space
    - E.g., instruction memory has 1024 words, data memory has 1024 word

    → 0 ~ 1023 words for instructions and 1024 ~ 2047 for data

init_data_Placement

- **Specifying the content of memory and register**
- **This file is processed by void setup_data_memory() in machine.c**

ecns1.csie.ntu.edu.tw - PuTTY

```
    IW    init_data_placement.txt
# data placement has two fields: address and data
# for data memory the address must be a multiplication of 4
# for register_file, the address must start with a $ sign
# anything after # sign is comment
# addresses not specified --> no data in that address (initialized to 0)
# address       data
-data_memory
64        12
80        30
88        10

-register_file
$1        0
$2        5
$3        64
```

assembly_code.txt

- This file is processed by void setup_inst_memory() in machine.c
- The first instruction is labeled with “Start”
- The program ends when hitting “EXIT” label

ecns1.csie.ntu.edu.tw - PuTTY

```
    IW    intput_code.txt
# the given assembly code is defined here
# three fileds: memory address, label and instruction
# mem_addr       label    instruction
0        Start:   LW       $1, 0($2)
4                 LW       $3, 2($2)
8        Exit:
```