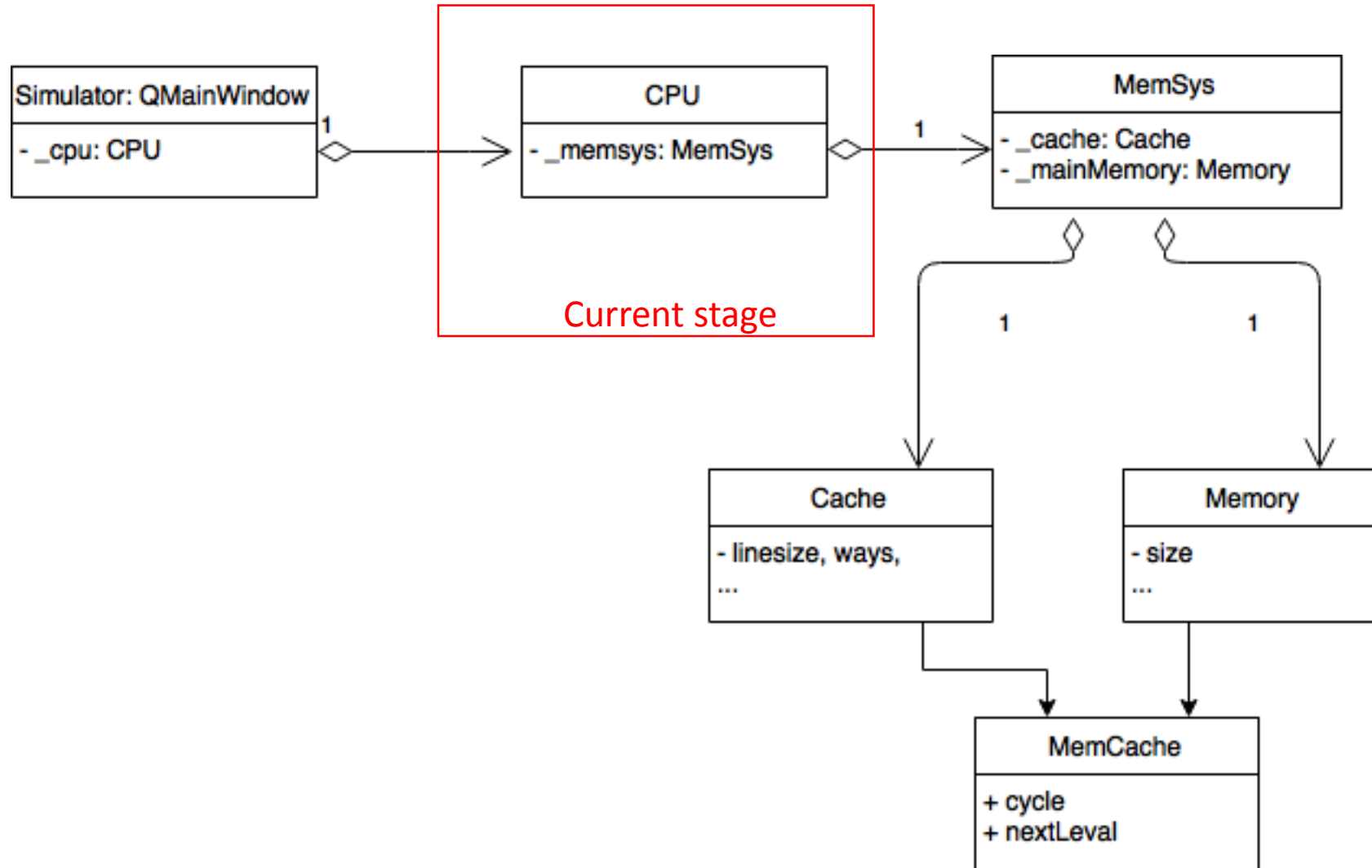# Pipelined basic instruction set simulation

Wei Hong and Tengyu Sun

Architecture

Implementation

- Instruction state

```
struct Instruction {
  int add;
  uint32_t npc;
  uint32_t ins;
  int type;
  int opcode;
  int rd1;
  uint32_t A;
  int rd2;
  uint32_t B;
  int rd3;
  uint32_t imm;
  int stage;
  uint32_t aluoutput;
  uint32_t lmd;
  bool cond;
};
```

- 5 stages

  IF -> ID -> EX -> MEM -> WB

- Pipeline

  represented as an instruction array of size 5. The instruction objects are passed around the 5 stages

- Registers

    16 general purpose registers
    16 floating point registers
    16 vector registers
    One program counter

```
class CPU {
public:
  uint64_t clk;
  CPU(MemSys* memsys);
  void run();
  void step();

private:
  uint32_t gpr[16];  //general purpose register
  float fpr[16];  //floating point register
  uint64_t vr[16];  //vector register
  uint32_t pc;
  uint32_t status;

  Instruction *pipe[5];
  bool err;
  bool clear;
  MemSys* _memsys;

  void ifc();
  void idc();
  void exc();
  void mem();
  void wbc();
};
```

Instruction supported

- Data transfer
  lb, sb

- Arithmetic and logical
  add,sub

- Control
  bgez, break

Table 1. Instruction Format				
Type 1	op offset			
opcode (7)	offset (25)			
Type 2	op $1,$2,immediate			
opcode (7)	$1 (4)	$2 (4)	immediate (17)	
Type 3	arithmetic and logical, sync, syscall,break			
opcode (7)	$1 (4)	$2 (4)	$3 (4)	

Basic assembler

- 1-pass

  Take assembly language clauses, parse into binary instruction

- A more general one is under development

  2-pass, support more types of instructions

Demo 1

preload mem[100] = 1, mem[101]=2

lb $0,$1,100 #$1 = 1

lb $0,$2,101 #$2 = 2

add $1,$2,$3 #$3 = $1 + $2 = 3

bgez $3,1 #$3>0, branch to sb...

lb $0,$3,100

sb $0,$3,102 #mem[102] = 3

break

Demo 2

preload mem[100] = 1, mem[101]=2

lb $0,$1,100 #$1 = 1

lb $0,$2,101 #$2 = 2

add $1,$2,$3 #$3 = $1 + $2 = 3

bgez $3,1 #$3>0, branch to sb...

lb $0,$3,100

sb $0,$3,102 #mem[102] = 3

break