Bridge of Life Education

Lab

Integrating Branch Predictor into Comet

Github Repository

- repo
- Directory:
  - README.md - The instructions of the whole workflow
  - lecture/ - The slides
  - lab/comet.py
  - lab/synthesizable/

Github Repository

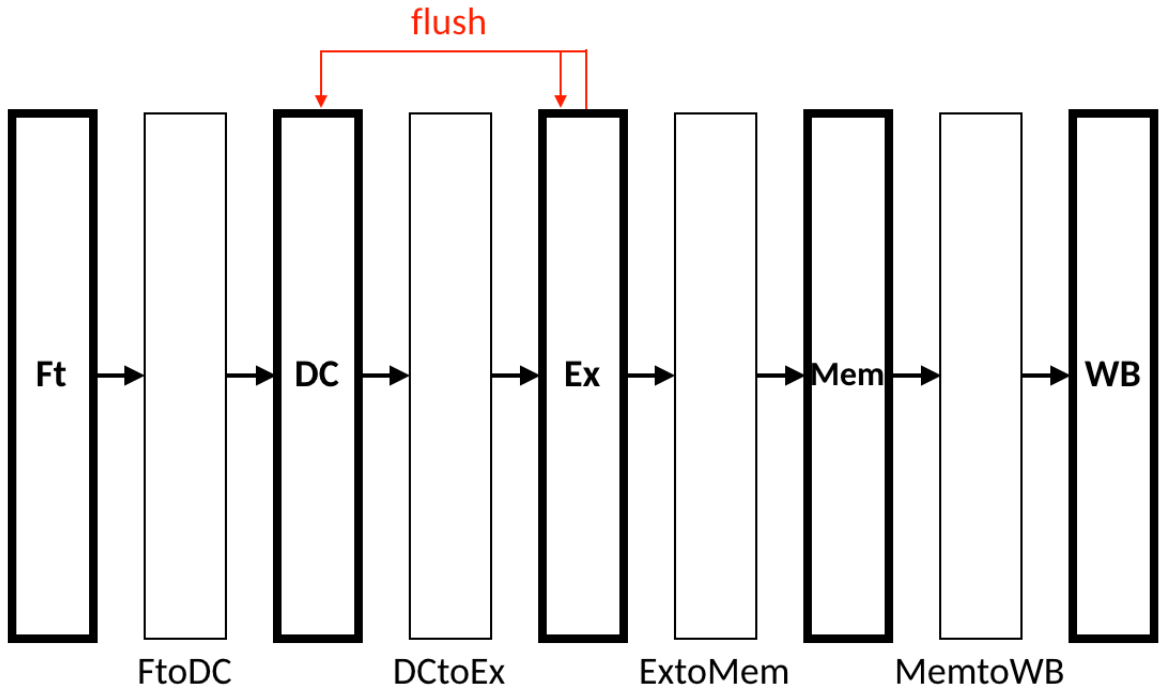
- lab/synthesizable/
  - benchmarks/
  - include/
  - src/
  - makefile
  - test.sh - To verify the design with all benchmarks

Branch Prediction

- In our basic design, a pipeline flush would be required to handle the branch, resulting in lower throughput
- If we can predict early in the pipeline whether a conditional branch instruction will actually cause a branch, we can branch to the correct address earlier, with less decrease in performance

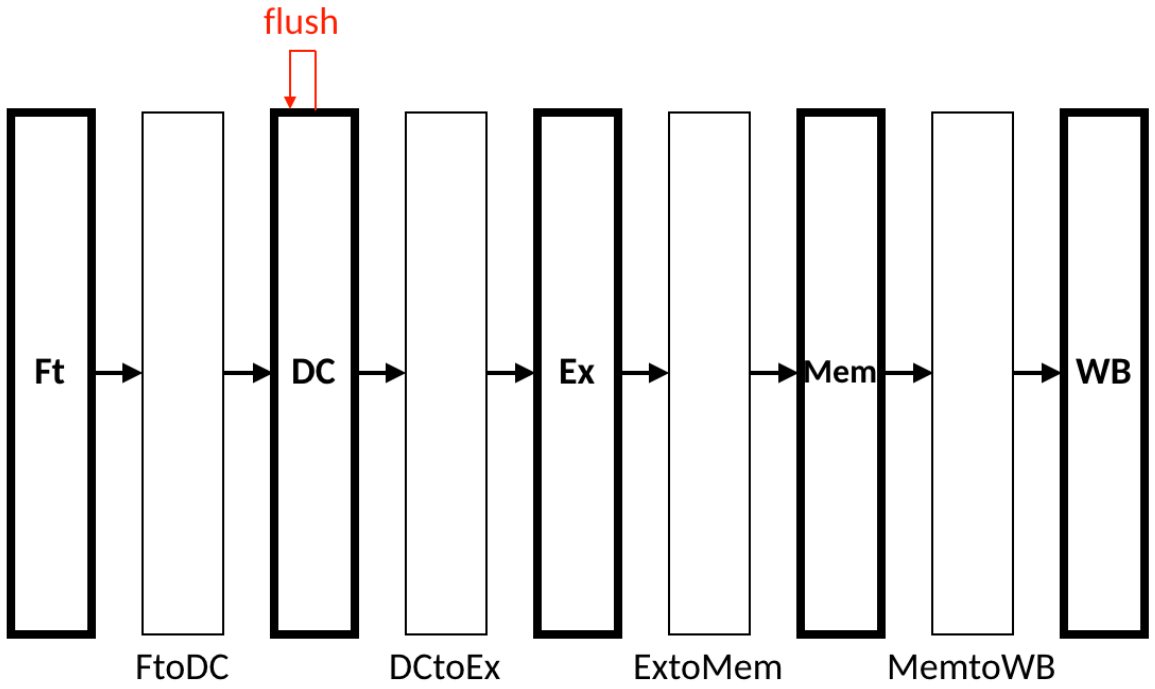
Branch Prediction

- Original design: flush 2 instructions



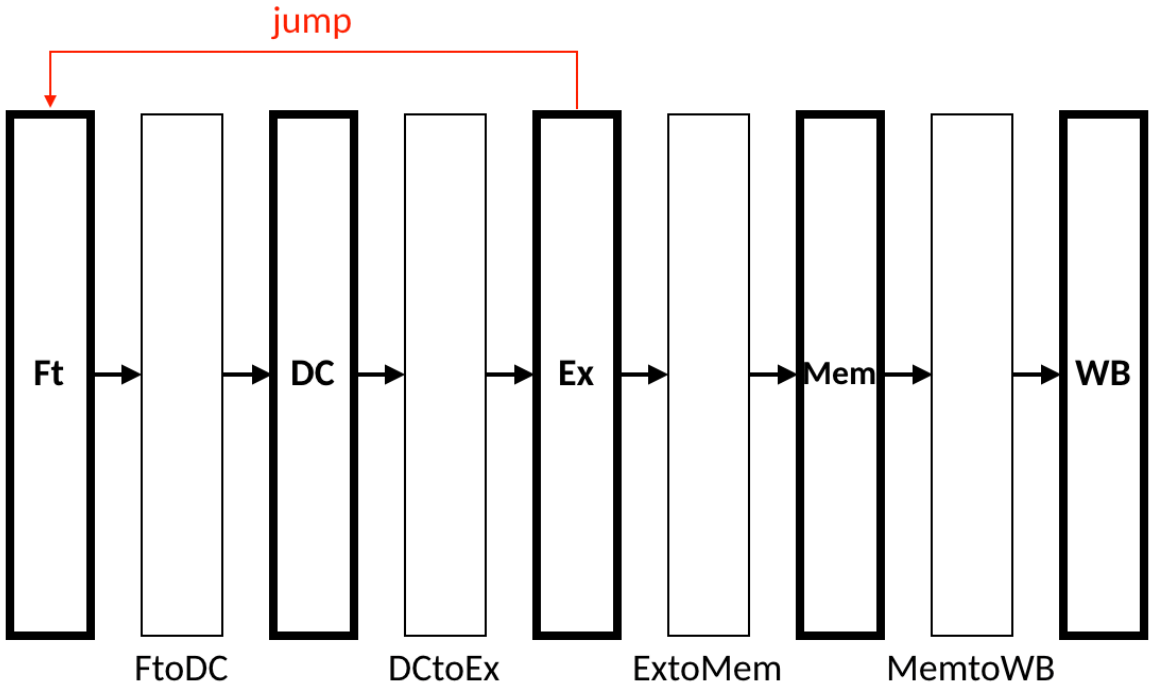
Branch Prediction

- With branch predictor: flush 1 instruction



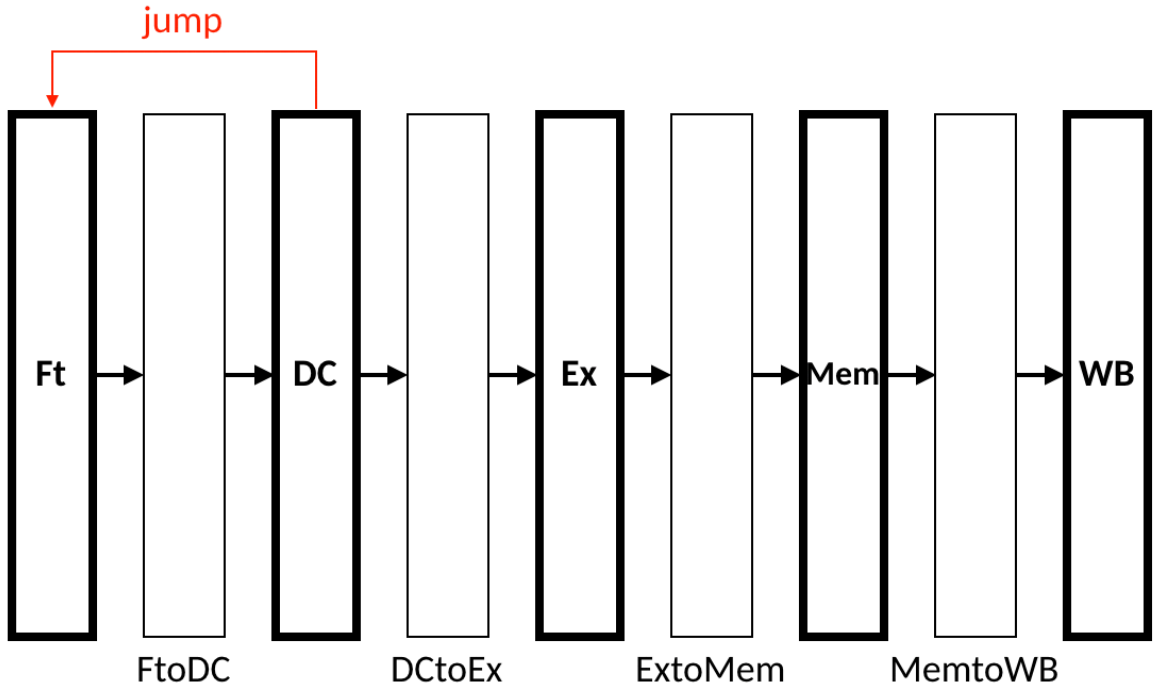
Branch Prediction

- Original design: calculate pc in Ex stage



Branch Prediction

- With branch predictor: calculate pc in DC stage



Branch Predictor

- Declared as C++ class, defined in `synthesizable/include/branchPredictor.h`

Branch Predictor

```
class NaivePredictor_1 {
public:
    CORE_UINT(32) record[2][2] = {0};
    NaivePredictor_1() {}
    CORE_UINT(1) _predict(CORE_UINT(32) pc) {
        return 1;
    }
    void _update(CORE_UINT(32) pc, CORE_UINT(1) predict, CORE_UINT(1) result) {
        record[predict][result]++;
    }
};
```

Branch Prediction

```
void DC(struct FtoDC ftoDC,
        struct DCtoEx* dctoEx,
        ...
        BranchPredictor* bp,
        ...)
```

Branch Prediction

```
if (/* is branch instruction */) {
    // the signals to the Ft() stage
    _dctoFt_branch_predict = // result of the _predict() function
    _dctoFt_branch_pc = // calculate the pc

    // the new member of dctoEx
    dctoEx->branch_predict = _dctoFt_branch_predict;
}
```

Branch Prediction

```
if (_extoDc_flush || _dctoDc_flush_out) {
    // now the flush signals may be sent from Ex or DC itself
    // flush
}
...
if (*prev_opCode == RISCV_LD &&
    (dctoEx->rs1 == *prev_dest || dctoEx->rs2 == *prev_dest)) {
    ...
    dctoEx->branch_predict = 0;
    _dctoFt_branch_predict = 0;
    _dctoFt_branch_pc = 0;
} else if (_dctoFt_branch_predict) {
    _dctoDc_flush_in = 1;
}
```

we can only flush pipeline when no pipeline stall is needed

Branch Prediction

```
void Ft(CORE_UINT(32) * pc,
        CORE_INT(32) ins_memory[MEM_SIZE / 4],
        struct FtoDC* ftoDC,
        CORE_UINT(1) _dctoFt_branch_predict,
        CORE_UINT(32) _dctoFt_branch_pc,
        CORE_UINT(1) _extoFt_jump,
        CORE_UINT(32) _extoFt_pc,
        CORE_UINT(1) _dctoFt_stall,
        CORE_INT(32) * insts) {
    CORE_UINT(32) next_pc = *pc;

    if (!_dctoFt_stall) next_pc += 4;
    *pc = // when should pc be set to _dctoFt_branch_pc?
    if (!_dctoFt_stall) {
        (ftoDC->instruction).SET_SLC(0, ins_memory[(*pc & 0x0FFFF) / 4]);
        ftoDC->pc = *pc;
        (*insts)++;
    }
}
```

Branch Prediction

```
_extoDc_branch_pc      // the pc of the instruction
_extoDc_branch_predict // the branch prediction (dctoEx.branch_predict)
_extoDc_branch_result  // the actual result computed by the Ex() stage
```

Branch Prediction

```
switch (dctoEx.opCode) {
    case RISCV_BR:
        ...
        extoMem->memValue = dctoEx.pc + dctoEx.datac;
        _extoFt_jump =      // when is a jump needed?
        _extoFt_pc =        // if a jump is needed, which pc should Ft jump to?
        _extoDc_flush =     // when is a flush needed in DC?
        _extoEx_flush_in = // when is a flush needed in Ex?
        _extoDc_branch_pc = dctoEx.pc;
        _extoDc_branch_predict = dctoEx.branch_predict;
        _extoDc_branch_result = extoMem->result;
        break;
    ...
}
```

Branch Prediction

```
if (_extoDc_branch_pc) {
    bp->_update(_extoDc_branch_pc,
                _extoDc_branch_predict,
                _extoDc_branch_result);
}
```

Performance Evaluation

- Simulation result of matmul16_16 (NaivePredictor<0>)

```
$ ./testbench.sim benchmarks/correct/matmul16_16/
exit at pc 67120.

clock cycle: 160342
instruction count: 130625
branch non-taken & predict non-taken: 291
branch non-taken & predict taken: 0
branch taken & predict non-taken: 4445
branch taken & predict taken: 0
pipeline stall: 29717
pipeline flush: 9490
```

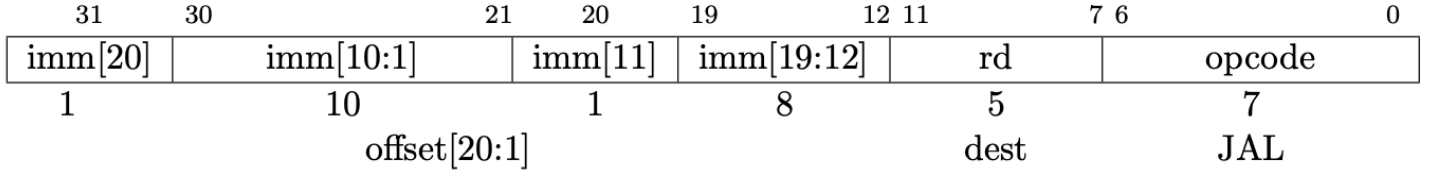
Performance Evaluation

- Simulation result of matmul16_16 (NaivePredictor<1>)

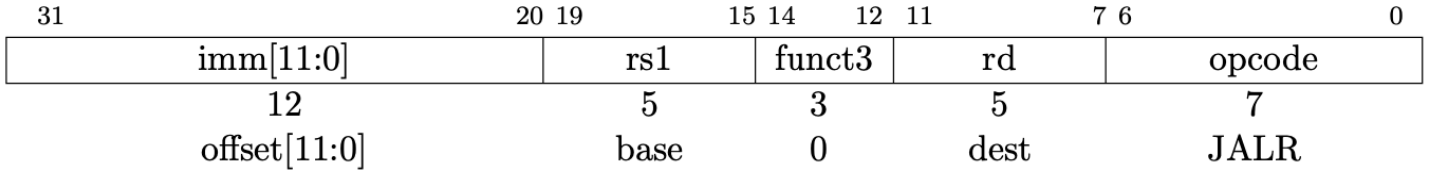
```
$ ./testbench.sim benchmarks/correct/matmul16_16/
exit at pc 67112.

clock cycle: 156477
instruction count: 126760
branch non-taken & predict non-taken: 0
branch non-taken & predict taken: 291
branch taken & predict non-taken: 0
branch taken & predict taken: 4445
pipeline stall: 29717
pipeline flush: 5918
```

Discussion

- Since the DC stage is able to flush the pipeline and jump to another pc now, can we also handle JAL instructions in the DC stage?



- How about JALR instructions?



Submission

- [Student ID]_lab-riscv.patch
  - git diff > [Student ID]_lab-riscv.patch