# Project 4

CSI3102-2, 2017 Computer Architecture

May 30, 2017

Project 4 에서는 gem5 를 이용하여 Out-of-Order execution 을 시뮬레이션합니다. 각 instruction 들의 issue/execute/commit 시점을 분석하여 수행 순서를 확인하고, microarchitectural parameter 를 변경하며 그 결과를 분석합니다.

`git clone` 을 이용하여 다음 저장소의 프로젝트를 가져오세요.
https://github.com/dependablecomputinglab/ca_proj4.git

```
git clone https://github.com/dependablecomputinglab/ca_proj4.git
cd ca_proj4
git checkout ca_proj_4
```

# 1 Out-of-Order 시뮬레이션

## 1.1 gem5 환경 설정

gem5 는 기존 프로젝트와 동일하게 ARM cpu 를 시뮬레이션 합니다. `scons` 를 이용하여 ARM 바이너리를 생성해 주세요.

```
scons ./build/ARM/gem5.debug
```

## 1.2 벤치마크 설정

1번 문제에서는 gem5 에 기본적으로 포함되어 있는 hello world 테스트 프로그램을 사용합니다. 다음 위치에 테스트 프로그램이 존재하는지 확인해 주세요.

```
tests/test-progs/hello/bin/arm/linux/hello
```

## 1.3 시뮬레이션 수행

아래의 O3 시뮬레이션 방법과 Debug flag 를 사용하는 방법을 참조하여, 시뮬레이션을 수행하고 분석에 필요한 정보를 출력하세요.

### 1.3.1 O3 시뮬레이션

gem5 를 통하여 Out-of-Order 로 실행합니다. se.py 의 옵션 중 `--cpu-type` 을 `arm_detailed` 로 설정하고, `--caches` 옵션을 사용합니다.

```
./build/ARM/<gem5_binary> configs/example/se.py --cpu-type=
    arm_detailed --caches -c <arm_binary>
```

### 1.3.2 Debug flag 사용

Debug flag에 대한 설명. -r 플래그에 대한 설명. Debug flag 옵션을 설정하여 원하는 정보를 출력합니다.

```
./build/ARM/<gem5_binary> --debug-flags=IEW,O3CPU,Decode,Fetch,
    Commit -r configs/example/se.py ...
```

`-r` 옵션을 통하여 화면에 출력되는 결과를 `m5out/simout` 에 저장하였으므로, 해당 파일을 열어 debug flag 가 잘 작동하는지 확인해 주세요.

### 1.4 제출 방법

1.2 와 1.3 의 내용에 따라서, `IEW, O3CPU, Decode, Fetch, Commit` debug flag 를 사용하여 hello world 테스트 프로그램을 수행하세요. 결과 파일을 **4-1-학번.txt** 로 저장하여 제출하세요.

## 2 Instruction 수행 순서 분석

1번 문제에서 얻은 로그 파일을 이용하여 각 instruction 들의 수행 순서를 분석합니다.

### 2.1 로그 형식과 Sequence Number (sn)

Debug flag 를 사용하여 출력된 로그는 다음과 같은 형식을 따릅니다.

```
<tick>: <flag name>: <message>
```

아래의 로그는 85500 tick 에서 `system.cpu.iew` flag 에 의해서 출력되는 메시지이며, `0x8160` 의 instruction 을 처리하고 있음을 나타냅니다.

```
85500: system.cpu.iew: Execute: Processing PC (0x8160=>0x8164)
    .(1=>2), [tid:0] [sn:7].
```

또한, 위 로그 중 `sn` 은 sequence number 를 의미합니다. Sequence number 는 instruction 이 하나 수행될 때마다 gem5 에서 부여하는 숫자입니다. 따라서 위 로그는 7번 instruction 을 처리하고 있음을 나타냅니다.

### 2.2 수행 순서 분석

시뮬레이션을 통해 얻은 로그를 분석하여 모든 instruction 들의 sequence number, instruction name, issue / execution complete / commit 시의 cycle 을 출력하세요. 각 열은 탭 (\t) 으로 구분하세요. 아래와 같은 형식이 되어야 합니다.

```
#sn,inst,issued,execution_completed,committed
1       ldr r12, [pc, #36]      162     164     282
2       mov fp, #0              162     163     282
3       ldr r1, [sp] #4         162     165     298
4       addi_uop sp, sp, #4     163     164     298
5       mov r2, sp              163     165     298
6       str r2, [sp, #-4]!      170     172     298
7       subi_uop sp, sp, #4     170     171     298
8       str r0, [sp, #-4]!      170     173     298
9       subi_uop sp, sp, #4     171     172     298
...
```

아래 사항을 주의하여 구현해 주세요.

- 표에 들어가는 cycle 은 tick 값을 500 으로 나누어 표시하세요.
- `proj_4` 디렉토리 안의 `exe_order.py` 파일은 파일을 읽어 기본적인 오류 처리를 하는 템플릿을 제공합니다. 다음과 같은 형식으로 실행이 가능합니다.

  ```
  ./exe_order.py <input_file_name> <output_file_name>
  ```

- 다른 언어로 프로그램을 작성해도 괜찮지만, 위의 수행 형식 (`<binary_name> <input_file_name> <output_file_name>`) 을 유지해야 합니다.

### 2.3 제출 방법

`exe_order` 디렉토리를 만든 후, 디렉토리 안에 사용한 소스를 넣어 주세요.

## 3 제출 방법

**4-1-학번.txt** 파일과 **exe_order** 디렉토리를 하나의 tar 만들어 제출하세요. tar 파일의 이름은 **project 학번.tar** 로 제출하세요.