Implement Report

- 개요
  - 기존 코드에서는 한 instuction을 받아 끝까지 실행 후, 다음 instruction 을 실행하는 방식이었다. 본 과제에서는 매 clock 마다 instruction 을 받아와, 각 stage 별로 control signal 을 분리하고 pipe line을 구축해 병렬적으로 실행하는 방식으로 바꾸어 보았다.
  - 기존 5개의 step 에서 instruction Fetch 와 Register read 를 합쳐 4개의 step으로 구성하였는데(step1~4), 이를위해 ARMCPU.v 파일과 ctrlSig.v 파일을 수정하였다.

ctrlSig.v

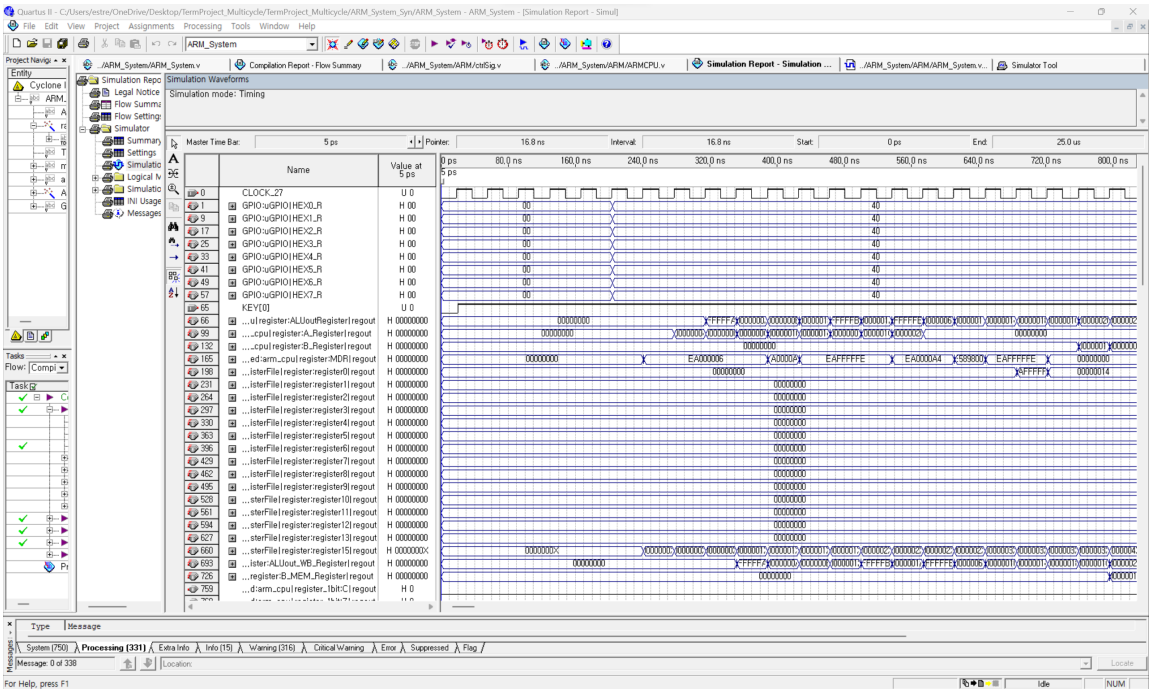
- **control signal pipeline 구현**
- `signalcontrol`
  - 기존 코드에서 s1 도 반환하는것으로 변경하였다.
  - stage 별로 control signal 을 분리
    1. step1 (IF, REG read)
       - `IRwrite, PCsave, immsrc, regbdst`
       - `PCsave` 는 BL instruction에서 reg[14] 에 기존 PC를 저장하는 플래그를 의미한다.
    2. step2 (ALU)
       - `ALUsrcA, ALUsrcB, ALUop, NZCV`
    3. step3(MEM)
       - `Mwrite, Mread`
    4. step4(WB)
       - `regwrite, regdst, regsrc`
  - bubble(stall) signal 추가
    - 모든 step에 대해서 아무것도 하지 않는다.
- `oneStep`

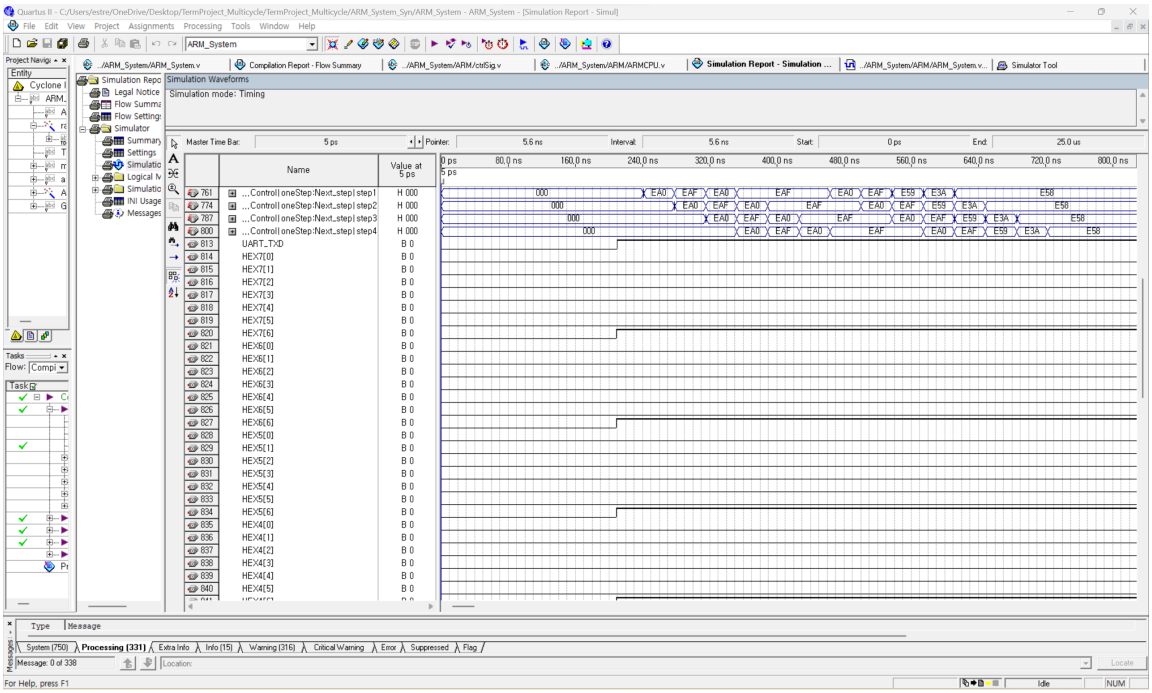
    - 새로 들어온 flags 를 step1 에 넣어준다.
    - 기존 step들은 다음 step으로 이동한다.
- `signalunit`
    - 각 stage(`staeg1~4`) 에 담겨있는 flags 에 따라, 필요한 signal을 가져온다.
    - 그리고 가져온 flags 는 기존 모듈처럼 반환한다.

ARMCPU.v

- `registerfile`
    - pc 에 관한 부분만 수정하였다.
    - `newPc` 를 받아 무조건 reg[15] 에 저장하고 반환한다.
    - `PCsv` flag가 켜져있으면,기존 pc(reg[15]) 를 (reg[14]) 에 저장한다.
- `armreduced`
    - pc
        - 기존 방식에서는 pc+8 , pc-4 로 최종적으로 step2에서 pc+4 가 reg[15]에 저장되는 방식이었지만, 매step 마다 pc+4를 바로 저장하는 방식으로 변경하였다.
        - `pc_next`를 `registerfile`에 넘겨서 pc를 업데이트 하는 방식인데, `pc_next`는 `setbubble` flag에 의해 결정된다.
            - `setbubble`==0 → 일반 instrucion 이므로 `pc_plus4`
            - `setbubble`==1 → Branch 관련 intruction `pc_branch`
        - 여기서 `setbubble` 은 IRwrite 에 의해 결정된다.
            - `IRwrite`==1 → 일반 instrucion 이므로 0
            - `IRwrite`==0 → Branch 관련 intruction 이므로 1
                - `setbubble`flag 를 1로 설정한다.
                - bubble instruction 이 들어가므로 한 step 동안 stall 하며 branch instruction 의 `ALUresult`를 기다릴 수 있다.
                - 다음클럭에서 `ALUresult`가 나오면 이를 `pc_branch`로 설정한다.
                - 이후 branch 명령어는 (MEM, WB) 단계에서 아무런 동작도 하지 않으므로, bubble 은 1개로 설정하였다.

        - `setbubble` 은 클럭이 한번 지나면 다시 0으로 줄어든다.
    - instruction
        - `setbubble` ==0 → inst
        - `setbubble` ==1 → 버블 instruction (32'b00000000000000000000000000000000) 를 넣어준다.
    - **data hazard 를 위한 data pipeline 구현**
        - 이전 step에서 나온 인스턴스 들을 다음 step 에서 사용해야하는데, 병렬실행 과정상 해당 인스턴스는 다른 instruction 이 사용하고 있다. 따라서 이를위해 register 모듈을 이용해 인스턴스 값을필요한 step까지 끌고가는 datapipe line 을 구현하였다.
        - `out2` : step1에서 나온다.
            - → `B` : step2 에서 `ALUnum2`로 사용된다.
            - → `B_MEM` : step3 에서 `writedata` 로 사용된다.
            - → `B_WB` : step4 에서 `regsrc`3로 사용된다.
        - `ALUresult` : step2 에서 나온다. (branch 의 경우 여기까지만 실행되고, 바로 next_pc 로 사용된다.)
            - → `ALUout` : step3 에서 `memaddr`로, step4 에서 `regsrc`2로 사용된다.
            - → `ALUout_WB` : step4 에서 `regsrc`1로 사용된다.
- 결과





- signal control 값은 순차적으로 의도한 대로 담기고, pc 도 clock 마다 업데이트 되는것을 확인하였다.
- 또한 step1~4에 flags 가 순차적으로로 담기며, 구현한 data pipeline 도 적절히 업데이트 되는것을 확인하였다.

- 한계
  - 다양한 Instruction 에 대해 의도한 대로 작동하는지는 확인하지 못하였다.