**Milestone4**

by kkh

#1

이번에도 어김없이 찾아온 milestone4 가이드입니다. 이걸 작성하는건 11/2일 새벽 1시네요. 여러분이 milestone1을 하고 계실 때죠. 저도 나름 지금 배우는 과목 숙제를 하다가 너무 모르겠어서 머리 좀 식힐 겸 짧은 가이드를 씁니다ㅋㅋ 이번부터는 디테일하게 알려드리진 않겠습니다. 3 정도 다 하셨으면 충분히 여러분께서 하실 힘이 있을 것이기 때문이죠 !

참고로 milestone4가 가장 시간 오래 걸리고 낙오자가 많이 생깁니다. 대부분 컴구조를 재수강 하실 분들이나 포기하실 분들은 여기서 갈리죠. 하지만 어려운 만큼 여태까지 배운 모든 것을 다 이해할 수 있는 기회입니다. 우선 문제부터 보고 넘어가봅시다. milestone4.pdf를 보세요.

#2  
문제가 말로는 겁나게 간단합니다.

Based on the Single-cycle CPU you have designed for the 3rd milestone, **implement the pipelining with data hazard detection and handling logic.**

milestone3이 끝나면 여러분은 jump, bne 같은 몇 몇 instruction을 실행할 수 있는 single cycle cpu를 만드셨을겁니다. 이제 이걸 기반으로 pipelining cpu로 만들고 여기서 발생하는 data hazard를 해결해야 합니다. 아마 굳이 보지 않아도 final-project\_milestone4.zip을 컴파일해서 dump 파일을 보면 pipeline 으로 실행할 때 data hazard가 겁나게 발생하는 프로그램이 분명할 겁니다. 얘네를 완벽하게 실행을 하는 CPU를 만들면 보드 7segment부분에 숫자 1, 2, 3, 4가 잘~ 실행이 될 거라고 하죠. 그럼 과제가 총 3개네요.

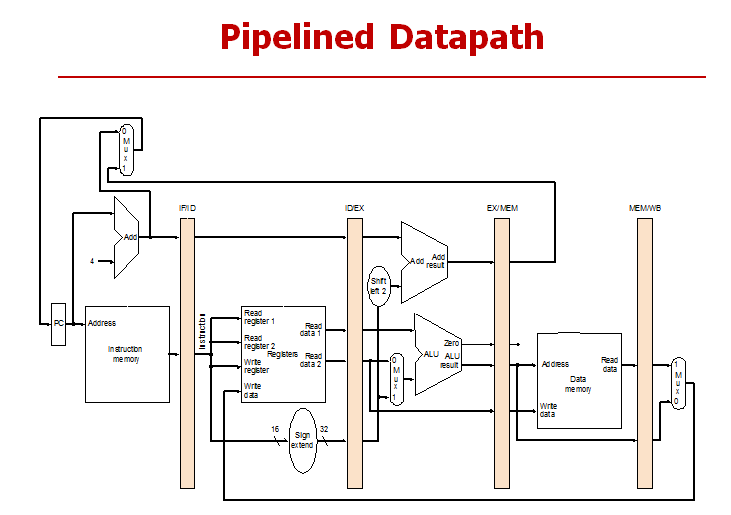
⓵ Single cycle cpu를 pipeline cpu로 만든다.

⓶ Data hazard를 해결하는 Forwarding을 구현한다.

⓷ Data hazard를 해결하는 Stalling을 구현한다.

이것만 하면 끝 입니다..

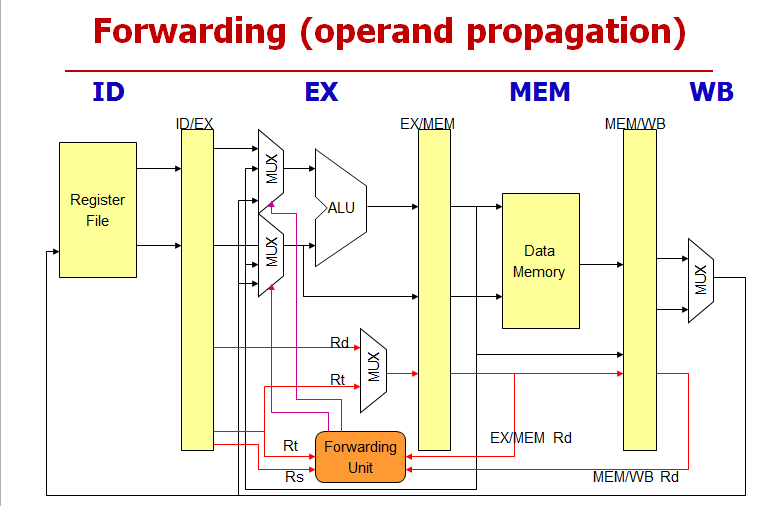
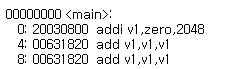
#3

**⓵ Single cycle cpu를 pipeline cpu로 만든다.**

이는 Lec5 Pipelined MIPS #2.ppt 보시면 앞부분에 아~주 자세하게 그림이 나와있습니다. milestone2를 잘 하셨다면 중간 중간 박아야되는 플립플롭 위치가 머리에 한 번에 그려질 것이라 믿습니다. 이 그림은 무지 간략화 해놓은 것이니깐 저것과 똑같이 그리신다면 안 될 것 같네요. 무튼 어느 정도 큰 도움이 되는 그림이긴 합니다. 플립플롭 크게 4개를 박으면 될 것 같구요, 그림에서 빠진 것이 module control 부분이 안보이네요. 얘가 어디에 위치해야할지 이런 것은 고민해보세요! control 모듈은 어디에 있어야하며 control signal들도 pipeline 5단계를 지나야 할 까~ 아니면 안 지나도 될까.. 뭐 이런 것부터 시작해서 말이죠.

무튼 이 부분 코딩하실 때 예전에도 강조 했지만 매우 신경 쓰셔야 할 부분이 wire선 정리입니다. 이젠 각 단계별로 정말 많이 wire들 이름도 바꾸고 잇고 해야할 텐데, 정리를 잘 해 놓으셔야 디자인하기 매우 편합니다. decode부분 ALU srca 부분이면 뭐 srca\_dec 이런식으로 분류를 잘 해 놓으세요.ㅋㅋ플립플롭 갖다 박는건 약간 노가다성에 꼼꼼함을 필요로 하는거니깐 이쯤 넘어가도록하죠.

**⓶ Data hazard를 해결하는 Forwarding을 구현한다.**

오른쪽 instruction 같은 전개가 data hazard가 발생하는 한 가지 경우죠. MemoryAccess stage(이하Mem)와 Execution stage(이하Ex) 사이에 발생하는 data hazard입니다. 하지만 이 경우만이 data hazard 일까요?

WB과 Ex도 충분히 발생 가능하죠?

**add v1, zero, 2048**

**nop**

**add v1, v1, v1**

이라고 생각해보세요. wb에서 ex로 forwarding 해줘야 할 것입니다.

역시 또 이것 뿐일까요? ID와....MEM?? WB??

ID와 MEM을 Forwarding 할 필요가 있을까요. 어차피 instruction 2개 차이라면 이건 WB과 Ex에 관해서 구현만 해놓으면 해결 할 수 있겠죠?? 그럼! 그림에는 안 나와 있지만 ID와 WB instruction 3개 차이가 날 때 forwarding을 해놓으면 될 것 같습니다! 기능 구현을 해줘야 하는 forwarding이 총 3개네요.

코딩은 없이 말로만 하고 있네요. 그럼 본격적으로 forwarding은 코딩은 어떻게하죠... 저같은 경우 작년에 ppt에 있는걸 거의 그대로 했습니다.. 근데 방법이 꽤 많더라구요. 그림처럼 꼭 안해도 되고요 당연히 여러분 자유이긴 합니다. 위에 ppt보시면 module ForwardingUnit을 만들었네요. ex stage의 Rs Rt와 다른 stage의 최종 destination인 exemem rd혹은 memwb rd가 같은 경우 alu에 들어가는 input을 바꿔줍니다. 모듈을 만드시는건 처음일테니 대충 디자인 해볼까요.

**module ForwardingUnit(인풋 rt, rs, exmem\_rd, memwb\_rd, 아웃풋 alumuxacontrol, alumuxbcontrol)**

**if (rs == exmemrd && \_\_\_\_\_\_\_\_\_\_\_\_)**

**alumuxacontrol = 2’b01**

**if else (rs == memwbrd && \_\_\_\_\_\_\_\_\_\_\_\_\_)**

**alumuxacontrol = 2‘b10**

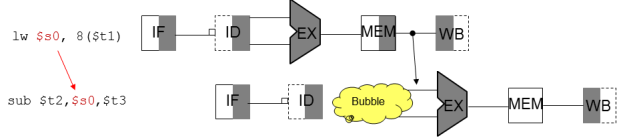
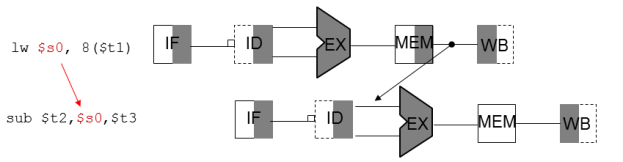
**else**

**alumuxacontrol = 2’b00**

이런식으로 쭈욱 코딩하시면 될겁니다. 그리고 모듈 만드시고 datapath안에다 불러서 wire이어 주면 되겠네요. \_\_\_\_\_\_\_\_\_부분엔 뭐가 들어가야 할지 한번 생각해보세요. 무조건 rd가 rs혹은 rt와 같다고 forwarding을 하는게 아니겠지요??

EMB000009942c68

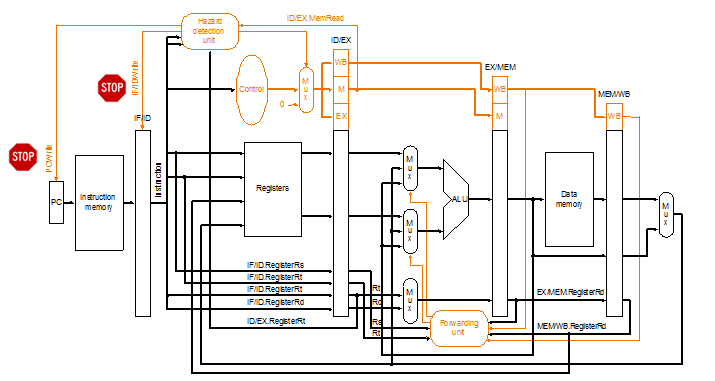
**⓷ Data hazard를 해결하는 Stalling을 구현한다.**



Data hazard는 forwarding만으로는 해결이 안되는 경우가 있죠. 바로 이런 경우입니다. lw라 함은 Mem stage에서야 원하는걸 가져오는데, 가져오고 있는 중에 dependency가 있는 그 다음 instruction이 execution에서 계산이 되고있죠.

EMB000009942c68번외로 사실 얼핏보면 forwarding으로 해결이 가능해 보이기도 합니다. 메모리에서 튀어나온걸 바로 execution stage에 넣어줘서 조금 늦게 계산 되더라도 1클락 안에만 된다면 굳이 stalling을 구현하지 않아도 되지 않느냐...그럼 performance손해도 없을텐데.. 작년에 있었던 질문인데요. 말이 안되는 겁니다. cache메모리 라는걸 들어보신 분이 있을겁니다. **메모리가 cpu입장에서는 너무 큰 거라서 한번 access한다는 자체가 시간이 엄청나게 걸려,** 따로 돈을 들여 조그만 메모리인 cache라는 걸 만들어 놓은겁니다. 예를 들면 고3 시절 책상에 책 갖다 놓는 것으로 비유해서 cpu가 책상이고, cache가 책상 옆에 박스 같은 간이 사물함, memory는 교실 맨 뒤에 진짜 큰 사물함. 그럼 뭐 하드디스크는 지하창고 정도 되는거죠. 다시 말해 하드 디스크에 access하는 것이 너무너무 오래걸려서 memory를 만들었는데 이것도 사실 access하는데는 너무 커서 cache라는걸 만든겁니다. 무튼 이 부분은 파이프라인이 끝나면 소름 돋을 정도로 많이 들을테니 요정도로 하고 넘어가겠습니다. 요점은 **메모리에 엑세스를 하는거 자체가 매우 시간이 오래걸려서 한클락 안에 왔다 갔다 할 수 있는게 아닙니다. 그리고 stalling이 없으면 syncronous한 cpu도 아닌 것 같습니다.** 우리는 장난감 cpu만드는거니깐 가능해 보이는 거죠.

무튼 stalling은 그림에서 bubble이라고 되있지만,

오른쪽 c2가 execution에서 mem stage로 넘어갈 때 다음 instruction 과 dependency가 있으면 그 뒤에 있는 30, 34 instruction이 플립플롭을 넘어가지 않고 stalling됩니다. 그리고 execution stage는 nop가 처리되도록 해주시면 되겠습니다. 그림대로 구현하신다면 완벽합니다. stop한다는건 pc와 fetch하는 플립플롭이 먹통이 되도록 해주시면 되겠죠? 아마 저 두 개의 플립플롭 손도 좀 보셔야할 것 같습니다. 아니면 mipsparts.v 여기 보셔서 적절한 플립플롭으로 바꿔주셔도 될 것 같구요. 여백이 좁으니 이것에 관한 코드는 생략하겠습니다.ㅎㅎ

#4

stalling 부분은 헛소리만 하고 설명이 끝났네요, forwarding하셨으면 이건 쉬울 거라고 생각합니다. 무튼 이 정도 하시고 modelsim 보는거 파이프라인 구조 생각하시면서 잘 보시면 그리 어렵지않게 끝내실 수 있을 것 같습니다. 화이팅요!!!