Computer Organization Assignment #2

MIPS ISS : Pipeline model with forwarding

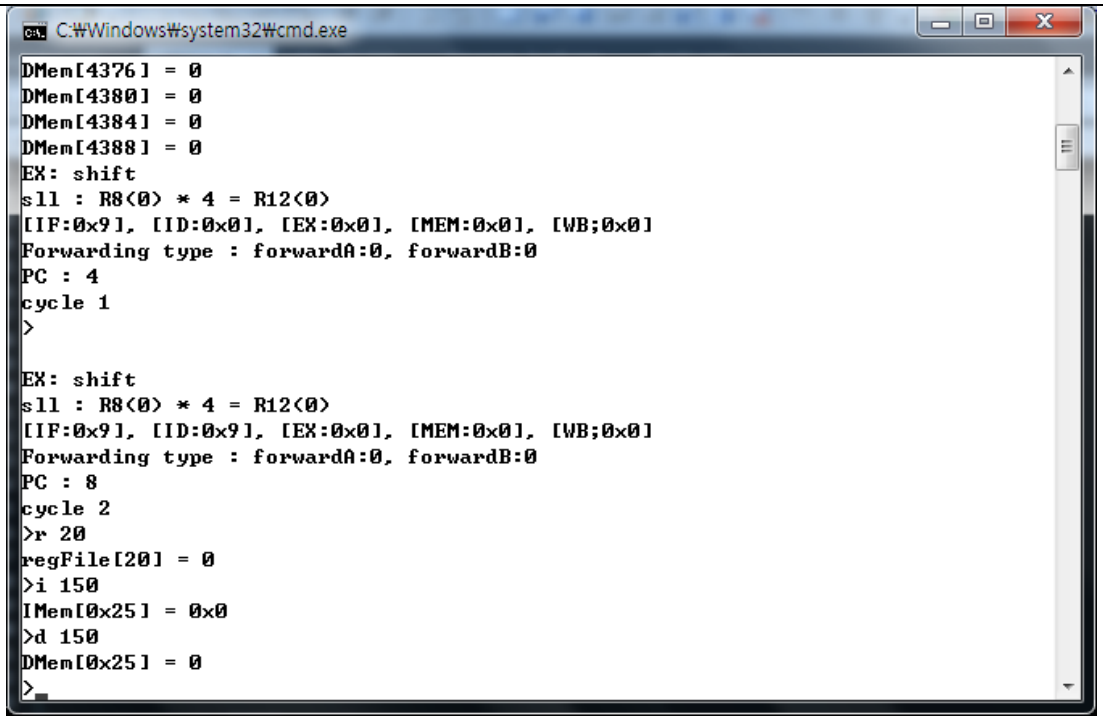
Due : 23:59, Wed, Nov 14, 2012

I. Introduction

C를 사용해 Pipeline 개념이 적용 된 ISS 모델을 설계 한다. 배포 된 C코드 프로젝트는 몇 가지의 Instruction에 대해서만 동작하게 되어 있으며, 5 stage pipeline을 위한 5개의 stage로 나누어져 있다. Assignment#1에서 미리 공부한 35개의 instruction set을 지원하는 ISS를 구현하고, 수업시간에 배운 hazard를 detect하여 forwarding 기법을 구현 한다.

II. Simulation Method

1. Visual studio 2010으로 생성 된 프로젝트가 기본적으로 제공 되며, 다른 컴파일러를 사용하는 경우에는 '₩MIPS_assignment2₩MIPS_assignment2₩source' 의 파일들을 모두 복사해서 나름대로 프로젝트를 만들어 실행하면 된다. (VS2010은 학교 소프트웨어 라이선스에 포함 됨) ※ '₩Assignment#2₩MIPS_assignment2₩MIPS_assignment2₩input_assm₩'의 '**imem.txt**' 파일을 입력으로 받아야 하므로 인풋 파일 경로지정을 해주어야 한다.
2. Visual studio를 사용하는 학생들은 ₩MIPS_assignment2₩MIPS_assignment2.sln 파일을 실행하면 솔루션이 실행 된다.
3. MIPSSim.h의 7line의 #define DEBUG를 주석 처리하면 키보드 입력 없이 시뮬레이션을 계속 진행한다. 반대로 DEBUG를 정의하면 각 cycle 마다 Instruction memory, Data memory, register의 정보를 알아볼 수 있다.



위의 그림에서 처럼 '**r 20**'을 입력하면 20번 레지스터 값이, **i 150**은 150번 iMem 값이, **d 150**은 150번 dMem 값이 출력 된다.

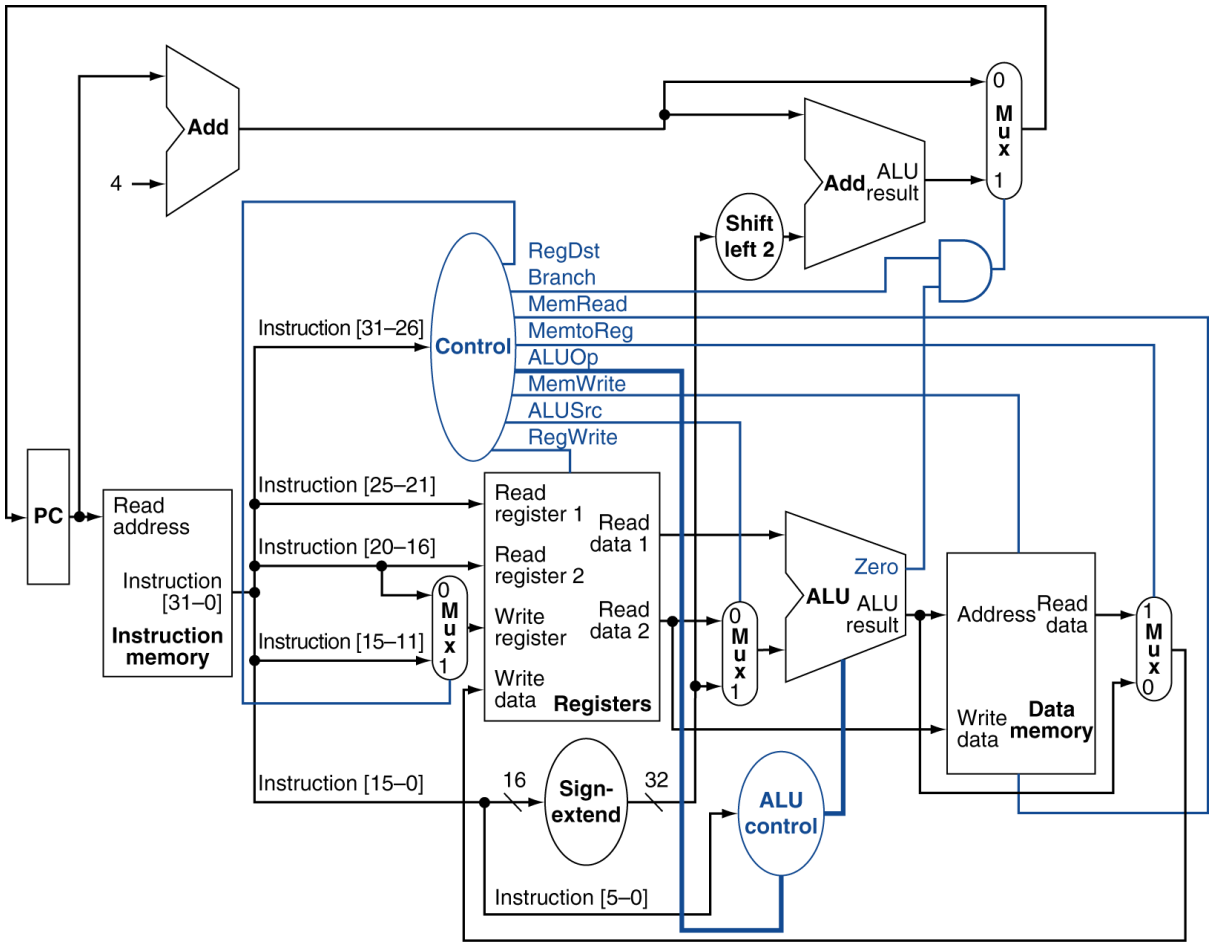
4. 각 소스파일의 정보

main.cpp	cycle을 한 번의 루프로 처리하는 전체 MIPS ISS 구조가 들어있음. 특히, pipeline register 업데이트나 control signal 및 각종 타이밍 문제는 여기서 처리해야 함.
MIPSsim.cpp	IF/ID/EX/MEM/WB 5개의 stage의 동작 function
MIPSsim.h	각종 define과 pipeline register의 data structure 존재함. pipeline register에 포함시키고 싶은 내용이나 수정하고 싶은 내용이 있을 시, 이 data structure를 수정해야 함.
program_load.cpp	메모리 초기화와 instruction을 파일로부터 읽어오는 function 모음.
command_line.cpp	디버깅을 위해 현재 상태를 출력해주는 명령을 받기 위한 function.

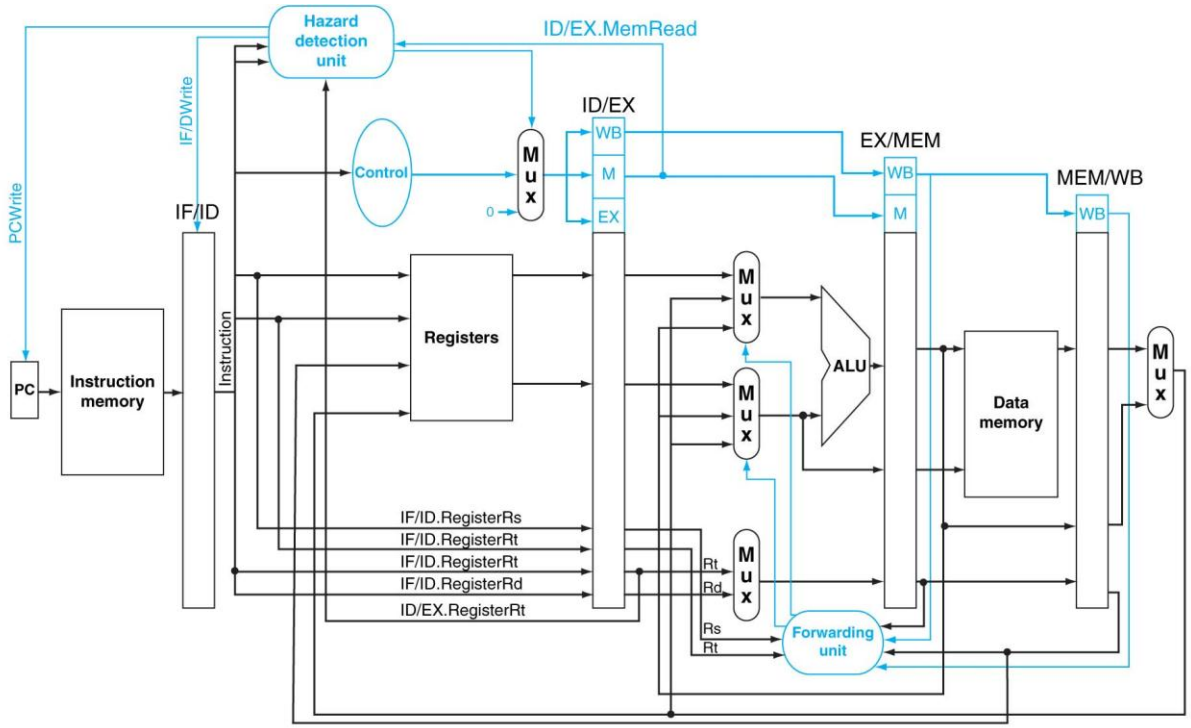
III. Requirements

아래 35개의 Instruction을 별첨의 'MIPS Instruction Reference' 을 참조하여 구현해야 한다.

ADD	ADDU	ADDI	ADDIU	SUB	SUBU
SLT	SLTU	SLTI	SLTIU	AND	ANDI
OR	ORI	XOR	XORI	NOR	SRL
SRA	SLL	SRLV	SRAV	SLLV	LW
SW	BEQ	BNE	BLTZ	BLEZ	BGTZ
BGEZ	LUI	J	JAL	JR	



(a) control signal 참조



(b) Pipelined control overview, showing the two multiplexors for forwarding, the hazard detection unit, and the forwarding unit

Figure 1 MIPS control signal and pipeline overview

Figure1을 참조해서 각각의 instruction의 control 신호를 만들어 주고 또한 forwarding을 위한 logic을 추가해야 할 것이다.

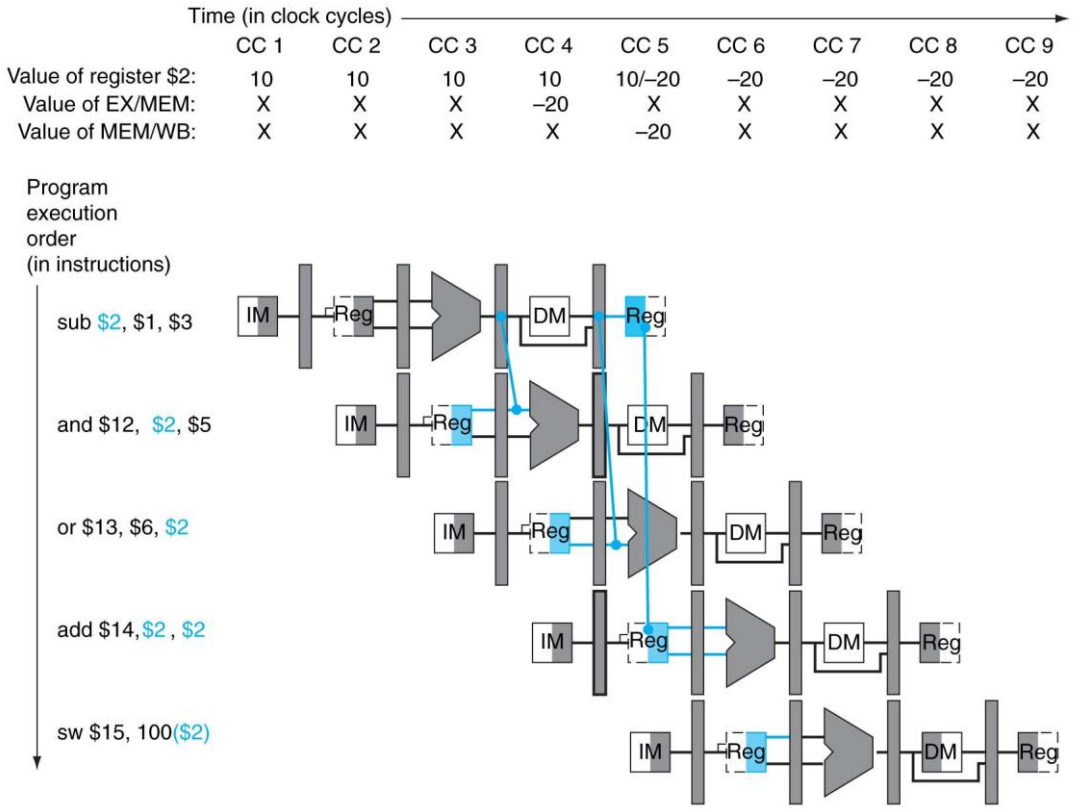


Figure 2 Data hazard & forwarding example

Figure 2는 instruction의 data 간 dependency에 의해 생긴 data hazard의 예를 보여 준다. 이전의 instruction의 결과가 그 다음 instruction에 사용 된다면 이전의 instruction의 결과가 나온 후에야 다음 instruction에서 그 결과를 받아서 사용할 수 있다. pipeline의 장점인 모든 stage가 쉬지 않고 일을 하게 하는 데는 성공하더라도 이렇게 hazard가 생겼을 경우를 시뮬레이터가 인식하여 필요한 stage에 정확한 데이터를 forwarding 해주거나, forwarding이 불가능한 경우에는 자동으로 stall(또는 flush)을 발생시키는 등의 방법으로 잘못된 데이터가 전달되지 않도록 해야 한다.

크게 다음의 3가지를 목표로 한다.

1. 35개의 instruction 구현.
2. hazard detection 및 forwarding logic 구현.
3. 메모리 read/write 동작을 10cycle 걸리도록 모델링 하고, 'bubble_sort' 프로그램을 돌려 프로그램 끝날 때까지 걸리는 cycle 수를 측정 한다.

IV. Implementation Details

1. 35개의 instruction 구현

ADD	ADDU	ADDI	ADDIU	SUB	SUBU
SLT	SLTU	SLTI	SLTIU	AND	ANDI
OR	ORI	XOR	XORI	NOR	SRL
SRA	SLL	SRLV	SRAV	SLLV	LW
SW	BEQ	BNE	BLTZ	BLEZ	BGTZ
BGEZ	LUI	J	JAL	JR	

assignment#1에서 조사했던 위의 35개의 instruction의 동작을 모두 구현한다. 몇 가지 대표적인 instruction에 대해서는 이미 구현되어 있다. 이를 위해 각 pipeline stage의 동작뿐만 아니라 control signal, pipeline register 업데이트 등을 점검해야 한다.

Requirement #1

35개의 instruction에 대한 동작 구현하고 구현 내용을 리포트에 설명 한다.

2. hazard detection unit 및 forwarding unit 설계

'₩MIPS_assignment2₩MIPS_assignment2₩input_assm₩' 의 imem.txt에 테스트용으로 제공하는 간단한 assembly code가 있다. 디버깅을 위해 'imem_for_mun_debug'를 참조하면 된다.

```
Instruction machine code      PC      Instruction
-------------------------------------------------------------
00100100000111010000001000000000  0       addiu $sp,$0,512
00100100000111100000001000000000  4       addiu $fp,$0,512
00000000000000000000000000000000  8       nop
00100111101111011111111111101000  12      addiu  $sp,$sp,-24
00000000000000000000000000000000  16      nop
00000000000000000000000000000000  20      nop
10101111101111100000000000010000  24      sw    $fp,16($sp)
00000011101000001111000000100001  28      move  $fp,$sp
00000000000000000000000000000000  32      nop
00000000000000000000000000000000  36      nop
10101111110000000000000000001000  40      sw    $0,8($fp)
10101111110000000000000000001100  44      sw    $0,12($fp)
10001111110000100000000000001000  48 L2 : lw    $2,8($fp)
00000000000000000000000000000000  52      nop
00000000000000000000000000000000  56      nop
00000000000000000000000000000000  60      nop
00101000010000100000000000001010  64      slt   $2,$2,10
00000000000000000000000000000000  68      nop
00000000000000000000000000000000  72      nop
00010000010000000000000000010111  76      beq   $2,$0,$L1
00000000000000000000000000000000  80      nop
00000000000000000000000000000000  84      nop
10001111110000110000000000001100  88      lw    $3,12($fp)
10001111110000100000000000001000  92      lw    $2,8($fp)
00000000000000000000000000000000  96      nop
00000000000000000000000000000000  100     nop
00000000000000000000000000000000  104     nop
00000000011000100001000000100001  108     addu  $2,$3,$2
00000000000000000000000000000000  112     nop
00000000000000000000000000000000  116     nop
10101111110000100000000000001100  120     sw    $2,12($fp)
10001111110000100000000000001000  124     lw    $2,8($fp)
00000000000000000000000000000000  128     nop
00000000000000000000000000000000  132     nop
00000000000000000000000000000000  136     nop
00100100010000100000000000000001  140     addiu $2,$2,1
00000000000000000000000000000000  144     nop
00000000000000000000000000000000  148     nop
10101111110000100000000000001000  152     sw    $2,8($fp)
00010000000000001111111111100100  156     b $L2
00000000000000000000000000000000  160     nop
00000000000000000000000000000000  164     nop
00000011110000001110100000100001  168 L1 :move  $sp,$fp
10001111101111100000000000010000  172     lw    $fp,16($sp)
00100111101111010000000000011000  176     addiu $sp,$sp,24
00000011111000000000000000001000  180     jr    $31
```

```
1 void main(){
2
3     int i = 0;
4     int j = 0;
5
6     for(i = 0; i < 10; i++)
7         j += i;
8 }
```

assembly code 의 원본 c code

앞의 그림은 0부터 9까지 합을 구하는 c program의 assembly code이다. hazard가 있을 수 있는 지점마다 충분한 길이의 nop을 삽입했다. (최적의 nop 개수가 아님) 주어진 디자인을 그대로 시뮬레이션 하면 dmem의 어떤 위치에 최종 결과값 45가 기록 된다. 이와 같은 결과가 나오면서 nop을 최대한 줄이는 방법으로 data hazard의 모든 경우를 detecting 하여 forwarding logic을 추가한다.

제공된 디자인에는 forwarding logic이나 hazard detection 기능이 없기 때문에 nop이 삽입되지 않은 코드를 입력하면 제대로 동작하지 않는다. 레지스터에 값이 겹치거나 메모리의 엉뚱한 주소의 데이터를 읽고 쓰거나 하는 등 datapath가 완전히 무너진다.

주어진 assembly code에서 nop을 완전히 제거하는 것이 목표이다. 경우에 따라 nop이 필요한 경우도 있는데(load-use data hazard) 이런 경우를 디텍팅 하여 flush 등의 방법으로 자체적으로 nop을 구현하면 된다.(HINT:stage register를 활용하라!)

Requirement #2

1. forwarding logic 구현하여 imem.txt의 nop을 모두 제거했을 때 프로그램이 정상 동작하도록 한다.
2. branch bubble, load-use hazard 등 nop이 강제로 필요한 경우에는 이를 detect하여 자체적으로 bubble을 만들어주는 hazard detect 기능 구현.
3. imem.txt의 최종 결과 45가 dmem의 어디에 기록되었는지 보이고, 총 cycle을 보고서에 기술 한다.

3. Bubble sorting program simulation

'₩MIPS_assignment2₩MIPS_assignment2₩input_assm₩requirement3_example'에 bubble sorting에 대한 assembly code가 있다. 각각의 instruction에 대한 구현은 끝났다고 가정하고 전체 프로그램이 흘러가는 것이 눈에 보이도록 16진수로 표기 했으므로 주의해서 iMem에 update 하도록 한다.

Requirement #3

1. memory read/write 동작을 수행할 때 10cycle의 delay가 생긴다고 가정하고 이를 구현하라.(현재는 MEM stage에서 1cycle만에 원하는 데이터가 메모리에 read/write 되는 구조)
2. Buble sorting 예제를 실행했을 때 끝나는 시점은 몇 cycle 인가? 총 몇 개의 instruction이 수행되었는지, 몇 cycle이 걸렸는지 체크할 수 있는 방법을 고안하여 코드에 적용하고 이 프로그램의 CPI를 구하고 설명하라.
3. 수행결과 dMEM 어떤 내용이 남으며 이는 iMEM에 저장 된 assembly 코드의 예상 결과와 일치하는가? 프로그램이 끝날 때 dMEM의 내용을 파일로 출력하여 제출하라. assembly 코드와 data memory 값 등을 총 동원하여 프로그램이 제대로 수행 되었음을 설명하라.

Additional Requirement : Exceptions

제공하는 디자인에는 디버깅을 위해 undefined opcode 등에 대해 assert() 를 사용해서 프로그램이 멈추도록 디자인 된 상태이다. 이에 대해 교과서에 나오는 flush, exception program counter(EPC), cause register, exception handler 등을 나름대로 정의 해서 simulation에 반영한다.

1. undefined op-code
2. ALU overflow
3. 정의한 memory range 벗어나는 주소로 접근 시도하는 경우

V. Submission

제출과제
1. 보고서(이메일과 hardcopy 모두 제출)
① Requirement#1 1) 35가지 instruction의 구현 내용을 보고서에 첨부하고 설명한다. ② Requirement#2 1) forwarding logic을 어떻게 구현했는지 보고서에 첨부하고 설명한다. 2) hazard detect를 어떻게 구현했는지 보고서에 첨부하고 설명한다. 3) imem.txt의 최종 결과 45가 dmem의 어디에 기록되었는지 보이고, 총 cycle을 보고서에 기술 한다. ③ Requirement#3 1) memory read/write가 10cycle delay를 갖도록 어떻게 구현했는지 설명한다. 2) bubble sorting 예제의 총 cycle과 CPI를 보인다. 3) bubble sorting 프로그램이 끝날 때 dMEM의 내용을 파일로 출력하여 출력파일을 제출하고, iMEM assembly code에 근거하여 제대로 된 결과가 나온 것인지 설명 한다. ④ Additional requirement 1) exception을 위해 구현한 사항들을 나열하여 보고서에 설명. 본인이 정의한 exception handler 등 모두 드러나도록 보고서에 기술.
2. 본인이 설계한 MIPS 모듈의 전체 프로젝트를 압축하여 첨부.
3. Requirement#3의 프로그램 수행 결과인 dMEM의 데이터를 txt 파일로 출력하여 제출.

제출방법
1. 조교 이메일 modesty@sdgroup.snu.ac.kr 로 보고서, 코드파일을 압축하여 제출. 2. 보고서 Hardcopy는 11월 15일(목) 수업시간 전에 제출
※ **11/14일 수요일 저녁 11시 59분 까지 제출분만 인정(메일 도착 시간 기준)** **(지각 제출은 받지 않겠습니다)**